A Flexible and Reusable Framework for Dialogue and Action Management in Multi-Party Discourse

Dissertation
zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten der Universität des Saarlandes

vorgelegt von

Markus Löckelt

Saarbrücken
2008

Tag des Kolloquiums: 26. Mai 2008

Dekan: Prof. Dr. Joachim Weickert

Mitglieder des Prüfungsausschusses:

- Prof. Dr. Thorsten Herfet *(Vorsitzender)*
- Prof. Dr. Dr. h. c. mult. Wolfgang Wahlster *(Erstgutachter)*
- Prof. Dr. Dietrich Klakow *(Zweitgutachter)*
- Dr. Jörg Baus *(Akademischer Beisitzer)*

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Diese Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Markus Löckelt
Saarbrücken, den 3. Juni 2008

Danksagung

Obwohl eine Dissertation natürlich nur von einer Person *geschrieben* wird, gibt es immer viele Personen, die Ideen, Diskussionen, Unterstützung und Aufmunterung zu ihrer Fertigstellung beitragen.

Zuerst will ich Professor Wolfgang Wahlster danken, der die Dissertation begleitet und mich mit konstruktiver Kritik und vielen Verbesserungsvorschlägen für die Entwürfe bedacht hat. Außerdem schuf er die Arbeitsumgebung, die die Dissertation überhaupt erst ermöglichte.

Ich danke auch Professor Dietrich Klakow, der sich bereiterklärt hat, die Rolle des Zweitgutachters zu übernehmen. Norbert Reithinger, der Leiter unserer Dialog-Arbeitsgruppe, verdient ebenfalls Dank: er hat mich immer gedrängt, es "endlich aufzuschreiben". Er war zudem eine unschätzbare erfahrene und freundliche Hilfe bei den Entwürfen, und auch während der Projektphasen selbst. Außerdem danke ich Professor Thorsten Herfet für die Übernahme des Vorsitzes des Prüfungsausschusses sowie Dr. Jörg Baus, der ebenfalls am Ausschuss teilnahm.

Wie gesagt, die Dissertation wäre nicht möglich gewesen ohne die anregende Arbeitsumgebung am DFKI. Das ist natürlich hauptsächlich den Leuten geschuldet, die man dort trifft. Ich will meinen Kollegen danken, mit denen ich an den Projekten *SmartKom*, *MIAMM*, *VirtualHuman* und *OMDIP* arbeiten konnte, in denen die hier präsentierten Ideen entstanden sind und realisiert wurden. Ich kann sie nicht alle erwähnen, aber namentlich seien genannt Norbert Pfleger, Elsa Pecourt, Tilman Becker, Jan Alexandersson, Gerd Herzog, Massimo Romanelli, Alexander Pfalzgraf, Alassane Ndiaye und Ralf Engel. Ich erfuhr auch überragende Unterstützung durch meine studentischen Hilfskräfte Ehsan Gholamsaghaee und Mohammed Mehdi Moniri, die sich manche Nacht um die Ohren geschlagen haben. Außerdem danke ich Kate Flynn, die den Entwurf als native speaker korrekturgelesen hat.

Schließlich danke ich für ihre nichtprofessionelle Unterstützung meinen Eltern, meinem Bruder und meiner Schwester, meinen Freunden und (last but definitely not least) Katja. Und für sie, und all die anderen, die die Aufgabe mit Geduld bis zum Schluß ertragen haben, will ich meine Danksagung mit einem treffenden Zitat aus einer anderen Publikation schließen: "Ich verspreche, daß ich *keine* weitere Dissertation schreiben werde" (Alexandersson, 2003).

(Hervorhebung im Original)

Acknowledgments

Although a dissertation is of course *written* by one person only, there are always many people that contribute ideas, discussions, support and encouragement to its completion.

First of all, I want to thank Professor Wolfgang Wahlster for supervising the thesis and providing me with constructive criticism and many suggestions for improvement on the drafts, especially in the final stages, and creating the work environment which made the thesis possible in the first place.

I also thank Professor Dietrich Klakow, who has agreed to take on the role of second advisor. The leader of our dialogue group, Norbert Reithinger, also deserves thanks: he was always pushing me to “write it down, already” and also was an invaluable experienced and friendly help in the drafting stages, and also during the project development itself. Additionally, I thank Professor Thorsten Herfet, who chaired the disputation committee, and Dr. Jörg Baus, who also participated in the committee.

As I said, this thesis would not have been possible without the stimulating environment at DFKI. This is of course mainly due to the people one meets there. I want to thank my collegues with whom I could work on the *SmartKom*, *MIAMM*, *VirtualHuman* and *OMDIP* projects, during which the ideas presented here were conceived and realized. I can not mention them all, but I'd like name Norbert Pfleger, Elsa Pecourt, Tilman Becker, Jan Alexandersson, Gerd Herzog, Massimo Romanelli, Alexander Pfalzgraf, Alassane Ndiaye, and Ralf Engel. I also received tremendous support from my student assistants, Ehsan Gholamsaghaee and Mohammed Mehdi Moniri, who worked tirelessly through many a night. In addition, I want to thank Kate Flynn for carefully proof-reading the draft as a native speaker.

Finally, for their non-professional support I thank my parents, brother and sister, my friends and (last but definitely not least) Katja. And for her, and all of the others who endured the quest with patience until completion, I want to close my acknowledgements with a fitting quote from another publication: “I promise that I will *never ever* write another thesis” (Alexandersson, 2003).

(Emphasis in the original)

Kurzzusammenfassung

Diese Arbeit beschreibt ein Modell für zielgesteuerte Dialog- und Ablaufsteuerung in Echtzeit für beliebig viele menschliche Konversationsteilnehmer und virtuelle Charaktere in multimodalen Dialogsystemen, sowie eine Softwareumgebung, die das Modell implementiert. Dabei werden zwei Genres betrachtet: Task-orientierte Systeme und interaktive Erzählungen. Das Modell basiert auf einer Repräsentation des Teilnehmerverhaltens auf drei hierarchischen Ebenen: Dialogakte, Dialogspiele und Aktivitäten. Dialogspiele erlauben es, soziale Konventionen und Obligationen auszunutzen, um die Dialoge grundlegend zu strukturieren. Die Interaktionen können unter Verwendung wiederkehrender elementarer Bausteine spezifiziert und programmtechnisch implementiert werden. Aus dem Zustand aktiver Dialogspiele werden Erwartungen an das zukünftige Verhalten der Dialogpartner abgeleitet, die beispielsweise für die Desambiguierung von Eingaben von Nutzen sein können. Die Wissensbasis des Systems ist in einem ontologischen Format definiert und ermöglicht individuelles Wissen und persönliche Merkmale für die Charaktere. Das *Conversational Behavior Generation Framework* implementiert das Modell. Es koordiniert eine Menge von Dialog-Engines (*CDE*s), wobei jedem Teilnehmer eine CDE zugeordet wird, die ihn repräsentiert. Die virtuellen Charaktere können autonom oder semi-autonom nach den Zielvorgaben eines externen Storymoduls agieren (*Narrative Mode*). Das Framework erlaubt die Kombination alternativer Spezifikationsarten für die Aktivitäten der virtuellen Charaktere (Implementierung in einer allgemeinen Programmiersprache, durch Planoperatoren oder in der für das Modell entwickelten Spezifikationssprache *Lisa*). Die Praxistauglichkeit des Frameworks wurde anhand der Realisierung dreier Systeme mit unterschiedlichen Zielsetzungen und Umfang erprobt und erwiesen.

Short Abstract

This thesis describes a model for goal-directed dialogue and activity control in real-time for multiple conversation participants that can be human users or virtual characters in multimodal dialogue systems and a framework implementing the model. It is concerned with two genres: task-oriented systems and interactive narratives. The model is based on a representation of participant behavior on three hierarchical levels: dialogue acts, dialogue games, and activities. Dialogue games allow to take advantage of social conventions and obligations to model the basic structure of dialogues. The interactions can be specified and implemented using reoccurring elementary building blocks. Expectations about future behavior of other participants are derived from the state of active dialogue games; this can be useful for, e. g., input disambiguation. The knowledge base of the system is defined in an ontological format and allows individual knowledge and personal traits for the characters. The *Conversational Behavior Generation Framework* implements the model. It coordinates a set of conversational dialogue engines (*CDE*s), where each participant is represented by one CDE. The virtual characters can act autonomously, or semi-autonomously follow goals assigned by an external story module (*Narrative Mode*). The framework allows combining alternative specification methods for the virtual characters' activities (implementation in a general-purpose programming language, by plan operators, or in the specification language *Lisa* that was developed for the model). The practical viability of the framework was tested and demonstrated via the realization of three systems with different purposes and scope.

Zusammenfassung

Diese Arbeit beschreibt das Ergebnis mehrjähriger Forschung und Entwicklung auf dem Gebiet des Dialogmanagements für multimodale Dialogsysteme. In mehreren Schritten wurde ein Ansatz, der anfangs im Kontext des multimodalen Einbenutzersystems *SmartKom* entstand, erweitert, um eine flexible Interaktion mit mehreren Benutzern in wechselnden Systemumgebungen mit unterschiedlichen Anforderungen und Funktionsumfang zu ermöglichen.

Der erste wesentliche Beitrag dieser Arbeit besteht in einem Modell für die Repräsentation zielgerichteter Interaktionen zwischen beliebig vielen menschlichen oder virtuellen Dialogpartnern in einem multimodalen Dialogsystem. Dabei werden zwei Typen von Systemen betrachtet: Task-orientierte Systeme zur Assistenz bei der Bewältigung von Aufgaben und interaktive Erzählungen (*interactive narratives*) im Bereich Unterhaltung und Edutainment. Speziell in letzterem Anwendungsszenario ist es dabei auch wesentlich, daß die vom System simulierten virtuellen Charaktere mit individuellen Charaktermerkmalen und Wissensbasen ausgestattet werden können.

Das CDE-Modell ordnet jedem der Dialogpartner eine eigene unabhängige Dialog-Engine (*Conversational Dialogue Engine*) zu. Es verfolgt einen modularen und inkrementellen Ansatz, der den Spezifikations- und Planungsaufwand durch die Verwendung wiederverwendbarer, zusammengesetzter Dialogbausteine – sogenannter *Dialogspiele* (Dialogue Games) – überschaubar zu halten sucht. Die Dialogspiele nutzen den Teilnehmern gemeinsame soziale und kommunikative Konventionen über die Dialogstruktur aus, um anhand bekannter und erprobter Muster einerseits die eigenen Aktionen in größeren Planungseinheiten konzipieren und andererseits Reaktionen von Dialogpartnern antizipieren zu können. Durch die Bekanntgabe der sich aus den Mustern ergebenden Erwartungen (*Expectations*) an die zukünftigen Reaktionen der Dialogpartner kann der Dialogmanager auch die mit der Analyse betrauten Module bei der Desambiguierung mehrdeutiger Benutzereingaben unterstützen.

Das Konzept der Dialogspiele in dem Modell kann auch benutzt werden, um “physische” Aktionen in der virtuellen Umgebung und Interaktionsprotokolle für anzubindende Anwendungsschnittstellen abzubilden. Das Modell nutzt weiterhin die Ausdrucksmächtigkeit und Flexibilität einer ontologischen Wissensrepräsentation. Dadurch können die Basis des Welt- und Dialogwissens in einer einheitlichen und standardkonformen Darstellung repräsentiert und unter Verwendung von generalisierten Werkzeugen wie dem *Protégé*-Editor definiert und bearbeitet werden. Es ist auch möglich, bereits existierende Basisontologien dem Modell anzupassen und wiederzuverwenden.

Der zweite Fokus der Arbeit ist der Entwurf und die Bereitstellung einer Programmumgebung *Conversational Behavior Generation Framework*, welche das CDE-Modell implementiert. Sie ist konzipiert für Dialogsysteme mit einer modularen und nebenläufigen Architektur und flexibel adaptierbar an vielfältige Systemanforderungen und Anwendungskonstellationen. In einem multimodalen System erlaubt sie in Kombination mit einer separaten Diskursmodellierungskomponente eine Interaktion mit komplex modellierten und parallel angesteuerten virtuellen Charakteren in Echtzeit. Jeder Charakter kann dabei entweder teilautonom nach Maßgabe einer externen narrativen Kontrollinstanz oder auch vollständig autonom agieren. Für den ersten Fall stellt die Umgebung ein Zielspezifikationsprotokoll (*directionML*) für das Verhalten der Charaktere bereit, welche einem externen Modul erlaubt, die Rolle eines “Regisseurs" zu übernehmen. Es kann dann den Fortgang der Geschichte dynamisch unter Berücksichtigung von Rückmeldungen über den Erfolg von Teilzielen anpassen (*Narrativer Modus*). Das Framework erlaubt auch planbasierte Aktionsplanung durch die Anbindung eines externen Planers sowie die Zusammenarbeit mit zusätzlichen applikationsspezifischen Modulen, wie beispielsweise dem in *VirtualHuman* eingesetzten *ALMA*-Modul zur dynamischen Modellierung emotionaler Zustände von virtuellen Charakteren.

Das Framework erlaubt die inkrementelle Entwicklung von Dialogsystemen durch die Verwendung von generalisierten Interaktionsbausteinen (building blocks) und den gleichzeitigen Einsatz unterschiedlicher Paradigmen für die Spezifikation von Anwendungen. Aktivitäten können definiert werden durch Code in einer allgemeinen Programmiersprache wie Java, durch Planoperatoren in PDDL, oder durch Aktivitätspläne in der Spezifikationssprache *Lisa*, die für die speziellen Anforderungen von Dialoganwendungen für das CDE-Modell entwickelt wurde. Es unterstützt Dateninspektion im laufenden System und eine manuell kontrollierte Ausführung von Teilzielen für Testzwecke während der Entwicklung. Um die Realzeitanforderungen in einem interaktiven System einzuhalten, wurde ein geschwindigkeitsoptimiertes API *JenaLite* zum effizienten Zugriff auf die ontologische Wissensrepräsentation entwickelt und eingesetzt.

Die Praxistauglichkeit der Umgebung wurde erprobt und erwiesen, indem sie zum Dialogmanagement in drei bezüglich Anforderungen, Szenario und Umfang sehr unterschiedlichen multimodalen Dialogsystemen, dem interaktiven Storytelling-System *VirtualHuman*, der taskorientierten Anwendung *OMDIP* und dem studentischen Projekt *Clue* eingesetzt wurde. Zum Abschluß der Arbeit wird vorgestellt, welche Besonderheiten diese Systeme aufweisen und wie das Framework eingesetzt wurde, um sie zu realisieren.

"If any such machines bore a resemblance to our bodies and imitated our actions as closely as possible for all practical purposes, we should still have two very certain means of recognizing that they were not real men.

The first is that they could never use words, or put together other signs, as we do in order to declare our thoughts to others [. . .] it is not conceivable that such a machine should produce different arrangements of words so as to give an appropriately meaningful answer to whatever is said in its presence, as the dullest of men can do.

Secondly, even though such machines might do some things as well as we do them, or perhaps even better, they would inevitably fail in others, which would reveal that they were not acting through understanding but only from the disposition of their organs. For whereas reason is a universal instrument which can be used in all kinds of situations, these organs need some particular disposition for each particular action; hence it is for all practical purposes impossible for a machine to have enough different organs to make it act in all contingencies of life in the way in which our reason makes us act."

— Descartes, Discourse 5

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Aims of this Thesis . . . 2
1.2.1 Problem Statement . . . 2
1.2.2 Thesis Contributions . . . 3
1.3 Summary of the Approach . . . 4
1.4 Preview of *VirtualHuman* . . . 5
1.4.1 The Football Quiz . . . 6
1.4.2 The Lineup Game . . . 8
1.5 Chapter Outline . . . 10
1.6 Notes to the Reader . . . 12

2 Requirements and Basic Concepts **13**
2.1 Introduction . . . 13
2.2 Characterization of the Task . . . 15
2.2.1 Task-Oriented Systems . . . 15
2.2.1.1 Task Structure . . . 16
2.2.1.2 Interaction Structure . . . 18
2.2.1.3 Purpose and Success Criteria . . . 19
2.2.2 Interactive Narratives . . . 21
2.2.2.1 Task Structure . . . 21
2.2.2.2 Interaction Structure . . . 23
2.2.2.3 Purpose and Success Criteria . . . 24
2.2.3 Related Types of Systems . . . 25
2.2.4 Summary . . . 26
2.3 The Interaction Triangle . . . 26
2.3.1 The Environment of the Interaction . . . 28
2.3.2 User Perspective . . . 29
2.3.2.1 Natural vs. Asymmetrical Interaction . . . 29
2.3.2.2 Response Delays and Feedback . . . 30
2.3.2.3 Believability and Immersion . . . 31
2.3.3 Dialogue Designer Perspective . . . 33
2.3.3.1 Defining the Knowledge Base . . . 33
2.3.3.2 External Control and Narration Engines . . . 33
2.3.3.3 Testing and Tuning the System . . . 34
2.4 Dialogue and Conversation Modeling . . . 34

2.4.1 Speech and Dialogue Acts 35
2.4.1.1 Types of Context Changes 36
2.4.1.2 Dialogue Acts 37
2.4.2 Discourse and Dialogue Structure 39
2.4.2.1 The Hierarchical Structure of Discourse Segments 39
2.4.2.2 Discourse History and Context 41
2.4.3 Information State 41
2.4.3.1 The Belief-Desire-Intention Model 42
2.4.3.2 *SharedPlans* 44
2.4.3.3 *TRINDI* Information State 45
2.4.3.4 *SmartKom*'s Information State 46
2.4.4 The Impact of Multimodality 47
2.4.4.1 Rationale 47
2.4.4.2 Processing 48
2.5 Dialogue Management 49
2.5.1 Patterns for Communicative Exchange 50
2.5.1.1 Dialogue Games 51
2.5.1.2 The Plan-Based Approach 55
2.5.1.3 Turn Management 57
2.5.1.4 Grounding 59
2.5.2 Multi-Party Dialogue 60
2.5.2.1 Speaker and Addressee Recognition 61
2.5.2.2 Participant Roles 62
2.5.2.3 Turn and Conversation Thread Management 62
2.5.3 Dialogue System Architecture 63
2.5.3.1 Order of Processing 63
2.5.3.2 Modularization and Communication 64
2.5.3.3 Module Types 65
2.5.4 Discussion 68
2.6 Summary 69

3 Related Systems **71**
3.1 Introduction 71
3.2 Task-Oriented Systems 71
3.2.1 *TRAINS* and *TRIPS* 72
3.2.2 *COLLAGEN* 73
3.2.3 *RavenClaw* 75
3.2.4 *WITAS* and other *TrindiKit*-Related Systems 76
3.2.5 *MATCH* 78
3.2.6 *QuickSet* 79
3.2.7 *SmartKom* 80
3.2.8 *SmartWeb* 83
3.3 Interactive Narratives 84
3.3.1 *Façade* 85
3.3.2 *The Sims* 86
3.3.3 *Mission Rehearsal Exercise* and *IN-TALE* 88
3.4 Other Systems and Approaches 90

3.5 Summary 92

4 Representing the Knowledge Base for Situated Conversational Characters 95
4.1 Introduction 95
4.2 The Role of Knowledge in a Dialogue System 96
4.2.1 Levels of Knowledge Representation 96
4.2.2 Types of Knowledge 97
4.2.3 Dialogue and Context 99
4.3 Ontological Representation 102
4.3.1 Representation Formalisms 102
4.3.2 Basic Ontological and Epistemological Terminology 103
4.3.3 Defining the Base Ontology 105
4.3.4 Mapping to an XML representation 106
4.3.5 Using the XML-Ontology as a Data Structure 109
4.4 Important Methods and Concepts 111
4.4.1 Unification, Restricted Unification and Overlay 111
4.4.2 Underspecification and Matching 113
4.4.3 Grounding 114
4.4.4 Relations and Condition Matching 116
4.4.5 Affordances 117
4.5 Summary 119

5 A Model for Generating Multi-Party Conversational Behavior 121
5.1 Introduction 121
5.2 Motivation and Basic Structure of the Model 122
5.2.1 Building Blocks 122
5.2.2 Layers of Action 122
5.3 The Dialogue System 123
5.3.1 The Multi-Party Conversation Manager 124
5.3.2 The Discourse Modeler 125
5.3.3 Communication With Other Modules: Channels 126
5.4 The Conversation Participants 127
5.4.1 Conversational Dialogue Engines 127
5.4.2 Perception Filters and Grounding 129
5.5 Interaction 130
5.5.1 Layers of Conversational Action 131
5.5.2 Motivation for Action 134
5.5.3 Action Modes 136
5.6 The Building Blocks of the Model 137
5.6.1 Act Types 139
5.6.1.1 Examples of Dialogue Acts 140
5.6.1.2 Physical Acts 142
5.6.1.3 Meta-Acts 143
5.6.2 Dialogue Game Types 144
5.6.3 Activity Types 148
5.7 Operation of the CDE Conversation Manager by Example 151
5.7.1 Tasks of the Conversation Manager 151

5.7.2 The Running Example from *VirtualHuman* 152
5.7.3 The Activities Available to a Character 153
5.7.3.1 Explicit Creation of an Activity Instance 155
5.7.3.2 Implicit Creation of an Activity Instance 156
5.7.4 Deliberation in Activities 157
5.7.5 Single-Initiative Dialogue Games 158
5.7.5.1 Directly Realizable Acts 159
5.7.5.2 Generated Acts 160
5.7.6 Consuming Acts 161
5.7.6.1 Determining the Processing Layer 161
5.7.6.2 Processing an Act on the Activity Layer 163
5.7.7 Multi-Initiative Dialogue Games 163
5.7.7.1 Starting an Intermediate Game 165
5.8 Summary 165

6 Realization of a Conversational Behavior Generation Framework 167
6.1 Introduction 167
6.2 CDE Framework Architecture 168
6.2.1 Overview 168
6.2.2 Accessing the Knowledge Representation: *JenaLite* 169
6.2.3 Configuration 169
6.2.3.1 Dialogue System Definition 169
6.2.3.2 Activity Specifications 172
6.2.4 CDE internal structure 175
6.2.5 The Framework's Graphical Interface 175
6.2.6 Connecting Modules and Devices to the Framework 178
6.3 Performing Activities and Dialogue Games 178
6.3.1 Goals 178
6.3.1.1 Setting Goals 178
6.3.1.2 Goal Feedback 179
6.3.2 Expectations 181
6.3.3 Activities 182
6.4 The *Lisa* Language 184
6.4.1 Motivation 184
6.4.2 The Structure of a *Lisa* Plan 186
6.4.3 The *Lisa* Interpreter 190
6.4.4 *Lisa* Language Elements 190
6.5 Using Planning to Schedule Activities and Games 191
6.5.1 Motivation and Applicability 191
6.5.2 Contingencies 192
6.5.3 Realization 193
6.6 Summary 193

7 Applications Implemented Using the Behavior Generation Framework 195
7.1 Introduction 195
7.2 *VirtualHuman* 196
7.2.1 Configuration 196

7.2.2 Narrative Mode and the Director CDE . . . 197
7.2.3 Character and User CDEs . . . 199
7.2.4 Realization Scheduling . . . 199
7.2.5 Affective Modeling in Cooperation with *ALMA* . . . 201
7.2.6 The Game Logic for the Lineup Game . . . 203
7.2.7 Dynamic Help and Explanations . . . 204
7.2.8 Timeout Reactions . . . 206
7.3 *OMDIP* and *Clue* . . . 206
7.3.1 *OMDIP* . . . 206
7.3.2 *Clue* . . . 208
7.4 Summary . . . 210

8 Conclusion 211
8.1 Contributions . . . 211
8.1.1 Scientific Results . . . 211
8.1.2 Practical Contributions . . . 213
8.1.3 Publications . . . 214
8.2 Future Work . . . 215

A *Lisa* Specification 217
A.1 Service Elements . . . 218
A.2 Slot Definitions . . . 218
A.3 Value Specifications . . . 219
A.3.1 Slots . . . 219
A.3.2 Relations . . . 220
A.3.3 Conditions . . . 221
A.4 Body Elements . . . 222
A.4.1 General Statements . . . 222
A.4.2 Assertions and Relations . . . 225
A.4.3 Control Structures . . . 227
A.5 Game Elements . . . 229

B Elements of the *JenaLite* API 231
B.1 Concept . . . 231
B.2 Interfaces . . . 232

C Schemata for DSD and *directionML* documents 237
C.1 The Dialogue System Definition . . . 237
C.2 The Direction Markup Language *directionML* . . . 239

D Communicative Acts 243

E Abbreviated XML Notation 245

Bibliography 247

List of Figures

1.1 Physical setup of the *VirtualHuman* scenario ZAMB 6
1.2 Phase 1 of the *VirtualHuman* ZAMB scenario 7
1.3 *VirtualHuman*: Sample interaction from the quiz phase 7
1.4 Phase 2 of the *VirtualHuman* ZAMB scenario 9
1.5 *VirtualHuman*: Sample interaction from the lineup game 9
1.6 Thesis outline 11

2.1 Complexity for various approaches to task-oriented dialogue 14
2.2 An example cinema reservation application 16
2.3 Example structure of a cinema reservation dialogue 18
2.4 *PARADISE* objective structure 20
2.5 The Aristotelian dramatic arc 22
2.6 Complexity progression of conversational systems 27
2.7 The Interaction Triangle 28
2.8 The virtual characters Miss Dewey, Mister Kaiser, and Smartakus 32
2.9 A taxonomy of dialogue acts 38
2.10 DSP relations in a fragment of a train ticket sale dialogue 40
2.11 Example multimodal context representation 41
2.12 Practical reasoning in a BDI agent 43
2.13 A *TRINDI* information state 45
2.14 An *InformationSearch* discourse object from a *SmartKom* information state . . 47
2.15 Examples of dialogue macrogames with joint goals 52
2.16 Example of dialogue games and their hierarchical structure 53
2.17 Example: An overhearer takes the role of responder 54
2.18 Notations for dialogue game combination operations 55
2.19 *STRIPS*-like action schemata for flight booking 56
2.20 A discourse actor algorithm 58
2.21 Examples for “barge-before” and “barge-in” 58
2.22 Grounding acts 59
2.23 Multi-party, multi-conversation dialogue layers 60
2.24 Addressee identification algorithm for multi-party interaction 61
2.25 Processing sequence for end-to-end multimodal dialogue processing 64
2.26 Sequential, direct and facilitator-mediated communication between modules . 65
2.27 Architecture for multimodal systems 66
2.28 List of module types 67
2.29 Modules in direct communication with the dialogue manager 68

3.1 *TRAINS* and *TRIPS* 72
3.2 *COLLAGEN* test application screen 74
3.3 *RavenClaw*: Hierarchy of agencies and agents 76
3.4 *WITAS* helicopter and system architecture 77
3.5 *MATCH* user interface and unimodal selection on the map 78
3.6 *QuickSet* simulation setup and facilitator 79
3.7 *SmartKom*: Platforms and screenshots 80
3.8 The *SmartKom* architecture 81
3.9 Internal structure of the the *SmartKom* action planner 82
3.10 Architecture of *SmartWeb* and *SmartWeb* on the MDA 83
3.11 Screenshots from *Façade* 85
3.12 Screenshots from *The Sims 2* 87
3.13 The *Mission Rehearsal Exercise* System 88
3.14 Screenshot from the *IN-TALE* prototype 89
3.15 Feature comparison of related systems 93

4.1 Classification of KR formalisms on several levels 96
4.2 Knowledge types and examples 97
4.3 Example for changes in different contexts effected by a dialogue act 101
4.4 Illustration of some ontological and epistemological terms 104
4.5 Base ontology structure 105
4.6 Editing ontological concept definitions in *Protégé* 106
4.7 Excerpt from the XML ontology representation 108
4.8 A *Menu* object represented in TFS notation 109
4.9 Some newly introduced subconcepts of *RepresentationalObject* 110
4.10 An example for unification 112
4.11 An underspecified *FootballPlayer* instance 113
4.12 Example for the grounding process 115
4.13 A *Relation* instance 116
4.14 A sample condition 117
4.15 An object with an affordance slot in *VirtualHuman* 118

5.1 Main information flow assumed for the dialogue system 124
5.2 The functional architecture of *FADE* 126
5.3 Conversation participants exchanging acts 129
5.4 Summary: ontology states and building blocks 131
5.5 Example process hierarchy in a sales dialogue 132
5.6 Different types of action motivations 134
5.7 Action cycle 136
5.8 Rhetorical question act type 138
5.9 Partial tree of subcategories of *Act* 140
5.10 Subconcepts of *MetaAct* 143
5.11 Some game types occurring in different contexts in *VirtualHuman* 144
5.12 Basic Question-Response dialogue game type 145
5.13 Instance of a dialogue game type corresponding to Figure 5.12. 146
5.14 Teacher question FSA 146
5.15 Illustration of the game composition operations 148

5.16 Some activity types in *VirtualHuman* 149
5.17 A scene from the quiz game 153
5.18 Excerpt from a quiz dialogue in *VirtualHuman* 154
5.19 *SetGoal* message from the narration engine 155
5.20 Some service mappings of the expert characters in *VirtualHuman* 156
5.21 A *FootballQuizQuestion* instance in the moderator's ontology 158
5.22 Outline of the moderator's "plan recipe" for a *QuizRound* 158
5.23 The "PresentVideo" game 159
5.24 Offering a selection of answers for the quiz question 160
5.25 Processing utterances from other conversation participants 161
5.26 The "QuestionSelection" dialogue game 163
5.27 The user *Agree*s with the opinion of expert Kaiser 165

6.1 Internal structure of a conversation manager instantiation 168
6.2 The set of knowledge resources for a CDE system 170
6.3 The DSD file for *OMDIP* 171
6.4 Excerpt from the activity definition file for the OMDIP system agent 173
6.5 The essential internal composition of a CDE 174
6.6 Ontology view for officer Bogert in the *Clue* system 176
6.7 The player preview showing a conversation in *VirtualHuman*. 177
6.8 Roles of the *SetGoal* concept 179
6.9 A *directionML* message setting the *Lineup* goal 180
6.10 Roles of the *GoalFeedback* concept 180
6.11 An example *GoalFeedback* message 181
6.12 An example *Expectation* containing a lexicon update 183
6.13 Process states 183
6.14 The activity manager loop 184
6.15 Distribution algorithm for the dispatcher 185
6.16 Activity Implementation Classes 186
6.17 Basic algorithm for a dialogue game's execution 187
6.18 The *ComeAgain* game from *OMDIP2* 188
6.19 Example *Lisa* activity specification, Part 1 188
6.20 Example *Lisa* activity specification, Part 2 189

7.1 The modules in *VirtualHuman* 196
7.2 Screenshot of the narration engine interface of *VirtualHuman* 198
7.3 Associated components of character and user CDEs in *VirtualHuman* 200
7.4 Realization time points of overlapping acts 201
7.5 The expert trying to grab the turn 202
7.6 The *CharacterModel* concept 203
7.7 Different affective states 203
7.8 A *FootballPlayer* instance from the *VirtualHuman* ontology 204
7.9 Requesting and accepting a move proposal by the expert 205
7.10 The *OMDIP* device and some sample screens from the application 207
7.11 The *OMDIP* architecture 208
7.12 Screenshots from *Clue* and module setup 209

A.1 Toplevel structure of a *Lisa* plan 217
A.2 XSD structure of *service* elements 218
A.3 XSD structure of a slot value specification 219
A.4 XSD structure of relation value specifications 220
A.5 XSD structure of a condition type 221
A.6 XSD structure showing the possible statements in a *body* 222
A.7 XSD structure of *subgame* statement 223
A.8 XSD structure of *lexiconUpdate* 224
A.9 XSD structure of *subgoal* 224
A.10 XSD structure of *try* 225
A.11 XSD structure of *setSlot* 225
A.12 XSD structure of *retractSlot* 225
A.13 XSD structure of *assertObject* 226
A.14 XSD structure of *setParameter* 227
A.15 XSD structure of relation operations 227
A.16 XSD structure of *select* 228
A.17 XSD structure of a conditional execution *loop* 228
A.18 XSD structure of a *game* type definition 229

B.1 *JenaLite* object hierarchy 231
B.2 The *Ontology* interface 233
B.3 The *OntoClass* interface 234
B.4 The *OntoSlotDefinition* interface 234
B.5 The *OntoInstance* interface 235
B.6 The *OntoSlot* interface 236

C.1 Schema for DSDs 238
C.2 Toplevel elements of *directionML* 240
C.3 The *goal* specification in *directionML* 241

D.1 Direct subcategories of *CommunicativeAct* 244

Chapter 1

Introduction

1.1 Motivation

The design of conversational user interfaces for interactions with computer systems is a broad and multifaceted field of research with great impact. Traditionally, to interact with a computer a user is either required to use formal means such as command line interfaces and computer languages, or to get acquainted with various sorts of *metaphors* that usually originate from other areas of life—desktops, files, menus, and so on—their meanings are not always intuitively clear, but have to be learned. In addition, those metaphors tend to break down in situations when it becomes obvious that, e. g., the "trash can" item of common computer GUIs behaves similar to, but not really the same as a ordinary trash can: For example, the contents of a real trash can do not just vanish when it is emptied. If the user is not aware of the cases where the metaphors do not apply, this can lead to problems. Another central point is that the purpose of a computer is to take care of, i. e., *perform*, tasks for and in cooperation with the user, which is something that is not commonly expected of inanimate objects like the ones mentioned, but something that would be more readily associated with a human counterpart offering a service. In this case, we do not "point an arrow" at or "click" on the fellow human, but we talk to each other using natural language, which may be accompanied by, e. g., pointing gestures and facial expressions. Several communication channels, or *modalities* used in combination constitute *multimodal communication*.

Enabling a computer to process and/or produce natural communication involving a combination of modalities is a difficult undertaking. Even arriving at the intended meaning of a multimodal communication act involves a large variety of sub-tasks, including recognition of the different analog modality inputs sensed by microphones, cameras, or other sensor devices, interpretation and fusion of related multimodal input pieces, and modeling dialogue history. On the other hand, the output requires the selection and conversion of computer data into human-intelligible units, presentation planning, generation, and synthesis of the different modalities. However, between understanding the user and constructing own multimodal utterances to the conversation from a given data representation, there remains the task of managing the interaction in a conversation, and actually doing the "job". To achieve this, it is not enough to just be able to handle the straight execution of simple commands, but rather involves participation in a conversational exchange of coherent two-way communication, and may include more than just “one human (commander) vs. one computer (executor)”, but multiple participants, in varying roles, on both sides; in other words, *multi-party conversation*.

In this thesis, we are concerned with the part of a dialogue system that is responsible for deciding what conversational actions the computer-controlled participants should do in a given situation, and triggering them. The functional part of our task is what is associated with the *dialogue manager*, or the *action planner*, of a dialogue system. It is the part of the conversational agents that *“controls the flow of the dialogue, deciding at a high level how the agent’s side of the conversation should proceed, what questions to ask or statements to make, and when to ask or make them”* (Jurafsky and Martin, 2000, p. 750). In systems like *VirtualHuman*, the behavior of the characters encompasses additional dimensions beyond pure speech interaction, such as emotional changes and expressions in other modalities. The dialogue manager described in this thesis does not directly generate this behavior, however, it triggers it (e. g., by updating an affect modeling module with cues on how current events are perceived emotionally). Therefore, it could more precisely be called a *conversational behavior generator for multiple characters*. To use a somewhat shorter name, we call it a *conversation manager* in this thesis. The term “dialogue” is also used in cases where the multi-party aspect is not essential, or where it is part of established terminology (e. g., “dialogue game”, “dialogue act”, or “dialogue system”).

The conversation manager relies on the semantic interpretation of the input provided by recognizer modules, and interacts with application and presentation modules to render an output for the user that represents its part of the conversation. This intra-system communication, hidden from the user, also represents actions intended, e. g., to retrieve information from a database to answer a question. Therefore, in addition to the procedural resources needed for action planning, we also consider the design and integration of the knowledge sources for the task, domain, and application models.

Some tasks commonly associated with the dialogue manager in a dialogue system are not covered by the approach. This includes processing of input to resolve ambiguities, reference resolution, and multimodal fusion. Another important such field is the generation of “subconscious” and reflexive behavior, such as turn-taking gestures and gazing behavior. For these tasks, it relies on another module, the *discourse modeler*. This role was taken over by the *FADE* module in the example systems (see (Pfleger, 2007)).

1.2 Aims of this Thesis

1.2.1 Problem Statement

The aim of this thesis is to provide a conversation management model and framework for dialogue systems that are able to let a group of participants—human users and virtual characters alike—engage in meaningful and purposeful conversation. It focuses on two types of systems, *task-oriented* dialogue systems and *interactive narratives*. For both cases, we examine the role of the conversation management component in the context of a functioning multimodal system with multiple modules.

The *VirtualHuman* project, which was realized with the framework, stages a virtual performance involving autonomous characters and allowing interaction from human users. During the project, two scenarios were developed for the performance. The earlier one is an edutainment scenario that involves two virtual characters, embodying a teacher and a pupil, respectively, enacting a lesson about astronomy with the human user taking the role of a second pupil. The other scenario is a competitive game about soccer for two human participants who interact with three different virtual characters. The interaction uses multiple input and output modalities. We will use examples from the second *VirtualHuman* scenario throughout the thesis for the sake of illustration.

1.2.2 Thesis Contributions

Because dialogue management is such an extensive and varied field of research, one thesis can only try to cover a very limited segment of it. The main contributions of this thesis are concerned with the following issues:

- **How can sophisticated, goal-oriented, and cooperative multi-party conversation be modeled?**

  We examine how to model conversations that involve an arbitrary number of mutually independent participants, where each participant can be either a human user or a virtual character. However, the conversations we look at are not free-form, but the participants find themselves in a particular situation and have individual goals which they pursue using communicative action. The purpose of the goals is to direct the behavior of the virtual characters in a way that is given by the dialogue designer. Goals can arise from the internal state of the participants themselves, or be imposed externally, e. g., incited by the behavior of other participants or by a designated software module—a so-called narration engine—that sets the goals for semi-autonomous characters.

  The model is meant to be suited for a variety of application scenarios. We aim for an approach that supports the construction of an application from building blocks that can be re-used and specialized if needed.

- **How to specify an adequate knowledge base for such a conversation**

  We aim for a knowledge base that is flexibly extendable and specified in a standardized representation. The representation must be sufficiently complete in expressive power to be used as a foundation for real-world application domains and it must be possible to implement the necessary inferences in a tractable way, i. e., the computations must be feasible in a system operating in real-time.

  The task should be made easier for the knowledge engineer by using a standardized formalism, such as an ontology. For ontologies, general tools are available that allow one to specify and maintain a knowledge base. The knowledge engineer should also not have to start from scratch for each new application, but be provided with a skeleton structure for the knowledge base, an upper ontology. The upper ontology already features a set of entities representing interaction building blocks that cover frequently used cases of dialogue acts, interaction patterns, and processes. This is meant to allow for an incremental construction of new applications with re-usable components.

- **How can the behavior of virtual characters be specified?**

  We want to realize believable and realistic characters that reflect individual character personality and other traits. There are several approaches to this question, with different advantages and shortcomings. We propose a method that allows the dialogue designer to combine different techniques in the same system, as long as they conform to the requirements of the dialogue model. This supports rapid prototyping by starting with a direct integration of procedures in a general programming language, and subsequently moving to a more abstract specification in an extensible behavior definition language, or as plan operators for an external planning algorithm.

- **Providing a unified and flexible framework that implements the model**

  We describe the actual implementation of our framework and show that it is indeed versatile enough to be used for "real-world" dialogue systems. To this end, we present how it was employed to realize three systems with quite different requirements: *VirtualHuman*, *OMDIP*, and *Clue*.

The focus of this thesis is to provide a framework that can be used as a practical tool for the construction of practical dialogue systems. It is not aiming to provide a comprehensive linguistic account for the dialogical interaction.

1.3 Summary of the Approach

The framework described in this thesis is a continuation of the work conducted on the action planning component of the DFKI "dialogue back-bone" toolbox that is intended to provide generalized modules covering all core functionalities for multimodal dialogue systems. Its development originated from the *SmartKom* project and was applied in several systems of varying size (Alexandersson et al., 2004a).

The action manager component is meant to be part of the implementation of a functioning framework, usable in a large variety of practical applications. This is already considered when we examine the background of previous work done in the area. There is a wealth of work in the field of dialogue management, nevertheless we need to focus on ideas that promise to be usable from a practical viewpoint, especially with regard to tractability from the perspective of computational and domain modeling expense. Numerous approaches, while theoretically sound and powerful, show weaknesses in the face of larger or more intricate domains, realistic real-time demands intended to make human-computer interaction convenient, or simply ultimate decidability of the computational problems involved.

This thesis builds on the available theoretical background to first define a model of conversation. This model in turn is used to construct a practical framework implementation intended to support realistic and user-friendly applications. The soundness and usability of the framework is demonstrated with several use cases of complete systems that were realized with it. We take a bottom-up course and start from the representation of knowledge. We then proceed through the steps, or levels, of how to model single acts in conversation, how to group related acts together in so-called dialogue games, and how to describe complex and coordinated activities for characters. Each subsequent level is built using the elements of the former. The levels are concerned with elementary dialogue acts, dialogue games, and goal-based complex activities. We have chosen an approach that models the necessary knowledge in an ontological representation. The characters are modeled as separate and autonomous entities, each using its own version of the ontology that encompasses the domain and task knowledge as well as the rules for conducting the dialogue. Additionally, the discourse history is recorded in a separate module using instances of concepts from the ontology.

Dialogue acts are seen as events in a virtual environment that trigger changes in the context maintained by the dialogue participants, such as providing new information, or imposing obligations on an addressee to act in a certain way. There is a clear separation between *public* and *private* information. Mutual and public agreement is assumed between the conversation partners about some effects that acts have (e. g., a question that imposes an obligation to answer to the addressee), but other aspects are hidden (e. g., whether the addressee actually honors the obligation, and why, as well as whether or not it chooses to answer honestly).

Dialogue games provide coherence structure and further describe what a specific dialogue act means in a situation, and with respect to the rest of the conversation. We use dialogue games not just as a means of analytically describing a given conversation structure, but also in a generative and procedural way, to plan, execute and coordinate the actions of characters to achieve their goals.

Composite goals (or tasks) that may require a number of related and ordered interactions until completion are represented by parameterized activities that the characters are capable of executing. The adoption of an activity can be triggered by means of actions of other conversation partners, by internal motivations of characters, or by a third party "direction" entity used to guide the structure of the interaction.

1.4 Preview of *VirtualHuman*

The framework described in this thesis is designed to be usable to realize a wide range of applications with different characteristics. Most of it was, however, developed in the context of the project *VirtualHuman*. We will use examples from *VirtualHuman* throughout the text for the sake of illustration; to put these examples into context for the reader, we give an outline here of what *VirtualHuman* is all about. Chapter 7 later fills in the gaps left by this introduction.

The realized system allows two human users to interact with up to three virtual characters at a time. The interaction exhibits *symmetric multimodality*, i. e., the users can use a combination of different modes of expression, and the virtual characters will respond accordingly. The *VirtualHuman* system itself was deployed using two different scenarios, the prototype "astronomy" scenario and the final "ZAMB" scenario.[1] In this thesis, we are only concerned with examples from ZAMB. ZAMB takes place in a 3D environment projected on a large screen, the users are placed in front of the projection behind two tables that feature microphones and

[1]The name ZAMB is a shorthand for "**Z**weiund**a**chtzig **M**illionen **B**undestrainer" (eighty-two million national coaches), a saying in German that plays humorously on the perceived opinion of many Germans that they know more about football than the national coach.

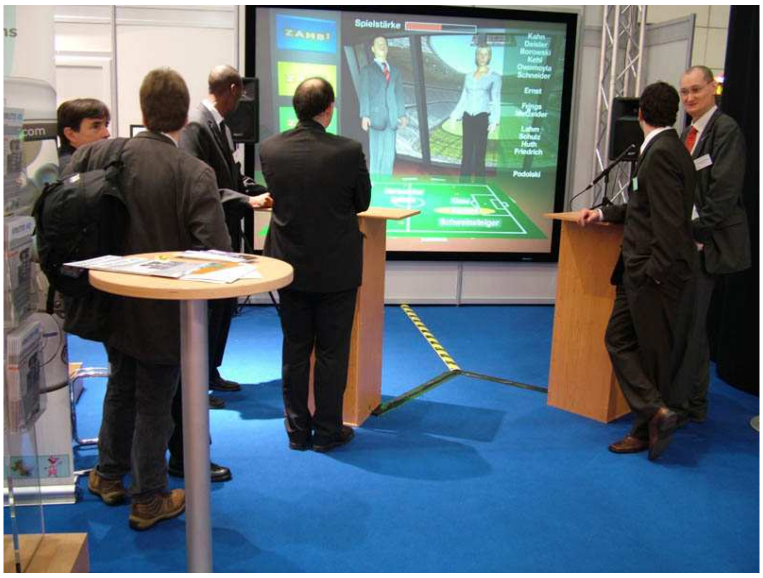


Figure 1.1: Physical setup of the *VirtualHuman* scenario ZAMB (as demonstrated at the CeBIT-2006 fair)

trackball interaction, as shown in Figure 1.1.[2]

The setting is modeled after a typical sports quiz show on television. It comprises two successive stages: The first stage, called the “football quiz”, allows two human users to compete on questions about sports events. It is hosted by a virtual moderator, who has two virtual experts at his side. Whichever user scores the most points in the quiz can continue to play in the second stage, the “lineup game”, which lets her create a lineup for a football team from a given roster of players. The overall duration of a complete ZAMB game is variable, and generally spans about fifteen minutes.[3]

1.4.1 The Football Quiz

The first stage of the scenario is designed for two human users interacting with three virtual characters. It is presented as in Figure 1.2. One virtual character, the *moderator* (on the left), hosts the show and gives instructions, while the two others—the *experts*, on the right side of the picture—take the roles of guest football professionals that offer to help the human users answer the quiz questions with their expert knowledge. The quiz takes place in a part of a virtual stage that features a video screen where key scenes from famous football matches can be displayed. The scenes are shown up to a decisive point (e. g., right before a goal shot). Then, the users are given a selection of possibilities about how the situation will develop (e. g., whether or not there will actually be a goal). The quiz has three rounds with a possible play-off round that is added if the competition is still undecided.

[2]It is also possible to use a different projection setup involving two overlayed projections, and have the users wear polarized glasses to achieve true 3D immersion.

[3]The maximum duration of the presentation can be configured dynamically.

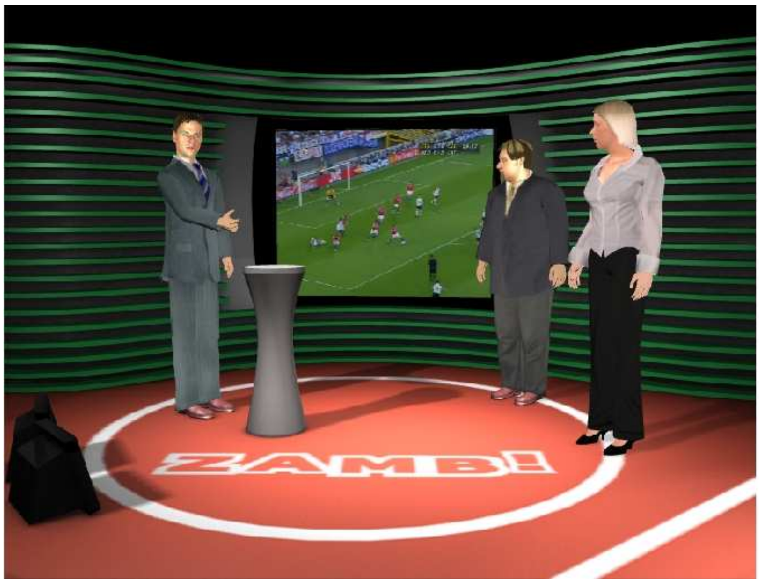


Figure 1.2: Phase 1 of the *VirtualHuman* ZAMB scenario: The moderator hosts the video quiz about football scenes shown on the screen in the back, the users can ask two experts for their opinion

(1)	Moderator:	. . . *Now look closely* [shows video on screen]. *What will happen next? The alternatives are* [counting gesture "1"] *One – Ballack scores the goal,* [counting gesture "2"] *Two – the keeper does a parade,* [counting gesture "3"] *Three – Ballack kicks the ball into the sky.*
(2)	Moderator:	*What do you think, Mister Kaiser?*
(3)	Expert Kaiser:	*I think Ballack scores the goal.*
(4)	Moderator:	[appreciative gesture] *Spoken like a real football trainer.*
(5)	Moderator:	[pointing to user 1] *Now, player one, what is your guess?*
(6)	User 2:	[tries to take over the role of user 1] *Answer one.*
(7)	Moderator:	*Well, actually I was asking player one.*
(8)	User 1:	*Miss Herzog, what do you think?*
(9)	Expert Herzog:	[blushes] *I think the keeper does a parade.*
(10)	Moderator:	*An interesting opinion.*
(11)	Moderator:	*Now it's your decision, player one.*
(12)	User 1:	*I think Mr. Kaiser is right.*
(13)	Expert Herzog:	[gets angry] *How can you believe this amateur!*
(14)	Expert Kaiser:	[smiles]
(15)	Moderator:	*Alright, answer one.*
[. . .]		

Figure 1.3: Sample interaction from the quiz phase

An example interaction from the quiz phase is shown in Figure 1.3[4]. Several interesting phenomena are present in this sample:

- The characters use and combine both “physical”—in the context of the virtual environment—and communicative actions,
- Utterances are multimodal; the speech is enriched by gestures that are precisely synchronized with the production of the spoken text. Some gestures derive from annotations of the utterances, others are generated automatically from the context (such as pointing at the addressee of an utterance). In turn (1), the counting gestures are generated explicitly by the questioning activity that enumerates the alternatives.
- The affective state of the characters changes in response to actions from the user as well as from the other virtual characters. The state is visualized in real-time by changing the facial expression and face textures of the characters, as well as their idle gestures and posture.
- There is conversation between the virtual characters as well as with the user. The user can use and refer to previous utterances from virtual characters in her own utterances.
- Characters overhear the conversation and react emotionally and verbally with respect to their attitude towards its content, even if they themselves are not addressed directly.

1.4.2 The Lineup Game

After the football quiz, the scenery on the stage changes with a camera movement, and the winning user proceeds to stage two, where she is challenged to put together a lineup for a German national football team against a given opponent with the help of the moderator and one of the experts.

Figure 1.4 shows this scene. Placed in front of the two active virtula characters, it shows a 3D football field representation, where players can be assigned to different playing positions by the user, at the front. On the right side is the team roster with the football players enlisted for the national team. A red bar on the top of the screen gives an evaluation of the current strength of the team (“Spielstärke”).

From this phase comes another example conversation (see figure 1.5). Some dialogue phenomena that are illustrated by this example, in addition to the ones mentioned in the previous section, are:

- The expert and moderator make comments about the moves of the player differently according to their role in the conversation and their different knowledge bases. The moderator prefers neutral acknowledgements, while the expert uses her knowledge to make more informed and evaluative comments (turns (3), (4), (8) and (9)).
- When more than one virtual character intends a contribution, they compete for the floor and indicate this by appropriate gesturing (turn (8)).

[4]*VirtualHuman* and the other systems described in Chapter 7 originally use German language; all conversation examples have been translated to English throughout this thesis.

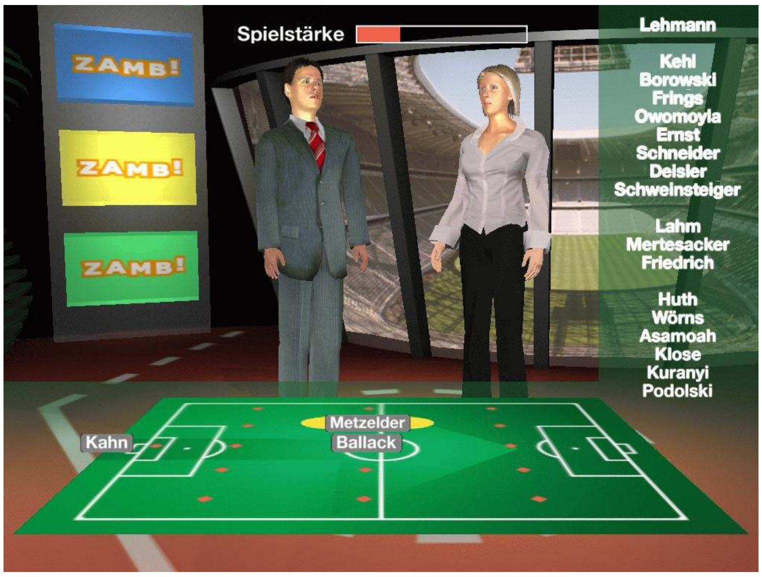


Figure 1.4: Phase 2 of the *VirtualHuman* ZAMB scenario: The moderator hosts the lineup game on a virtual football field; one expert offers advice

(1)	Moderator:	*Ok, let's get started.*
(2)	User:	*Put* [characters gaze at user] *Lehmann up as keeper.*
(3)	Expert Herzog:	[nods] *That's an excellent move! You can't go wrong with Jens Lehmann as a goalie.*
(4)	Moderator:	[nods] *Great, Lehmann as keeper.*
(5)	User:	*Miss* [characters gaze at user] *Herzog, give me a hint!*
(6)	Expert Herzog:	[smiles] *I recommend a defensive strategy against Brazil. I would definitely put Ballack into the central midfield.*
(7)	User:	*Ok,* [characters gaze at user] *let's do that.*
(8)	Moderator:	[nods] *Great, Ballack* [Expert makes turn-grabbing gesture] *as central midfielder.*
(9)	Expert Herzog:	[smiles, nods] *You won't regret this move.*
(10)	User:	. . . [hesitates]
(11)	Moderator:	[encouraging gesture] *Don't be shy!*
(12)	User:	*Hmm,* [characters gaze at user] *put Metzelder to Ballack's left.*
(13)	Moderator:	[shrugs] *That is not possible, I'm afraid that place is already occupied.*

[. . .]

Figure 1.5: Sample interaction from the lineup game

- The expert can use her knowledge to make a proposal based on dynamically planning the moves needed to transform the current lineup into an ideal team, or give general comments about attributes of the players based on her knowledge (turn (6)).
- The moderator follows the flow of the interaction and, according to a parameter in the set goal, will encourage the user to go on (turn (11)). This covers hesitations by the user as well as the overall time set for the game phase (by a narrative control instance) running out.
- the user can make spatial references that will be resolved according to the current configuration on the playing field, and the discourse history (turn (12)).
- There is a game logic that provides the moderator with an understanding of possible player placements at any given time. He will not allow the user, e. g., to put more than eleven players on the field or two players at the same position (turn (13)).

To be able to realize this kind of interaction, the scenario poses a sizable and multifaceted set of requirements on *VirtualHuman*'s system setup in general, and the conversation management component in particular. This thesis describes how the latter task was addressed.

1.5 Chapter Outline

The thesis comprises eight chapter parts, as shown in figure 1.6. The following enumeration briefly sketches the contents and purpose of each one:

1. This chapter, *Introduction*, gives a motivation identifying the problems we are concerned with, and addresses the question of why it is desirable to find solutions to them. It contains a short sketch of our approach, a short preview of the main application *VirtualHuman*, and finally this outline of what the chapters of the thesis are about.
2. The next chapter, *Requirements and Basic Concepts*, introduces the two different types of dialogue systems we are concerned with: task-oriented systems and interactive narratives. It gives criteria to judge whether a given system addresses its purpose successfully or not. It then highlights the different viewpoints of the system's users, the dialogue designer, and the designer of the system itself. The chapter also comprises sections on the rules of conversation, and the methods that are used in existing dialogue systems to devise, execute, and monitor behaviors for virtual characters. It also describes different architectural approaches for dialogue systems.
3. In the following chapter, *Related Systems*, we give examples of influential instances of dialogue systems with similar scope. It includes descriptions of task-oriented systems and interactive narratives that were realized. We show how these other systems treat the issues relevant to the tasks in this thesis.
4. The chapter *Representing the Knowledge Base for Situated Conversational Characters* describes how we address the crucial task of providing the characters with the different types of knowledge that are needed to engage in a conversation, and to solve the tasks

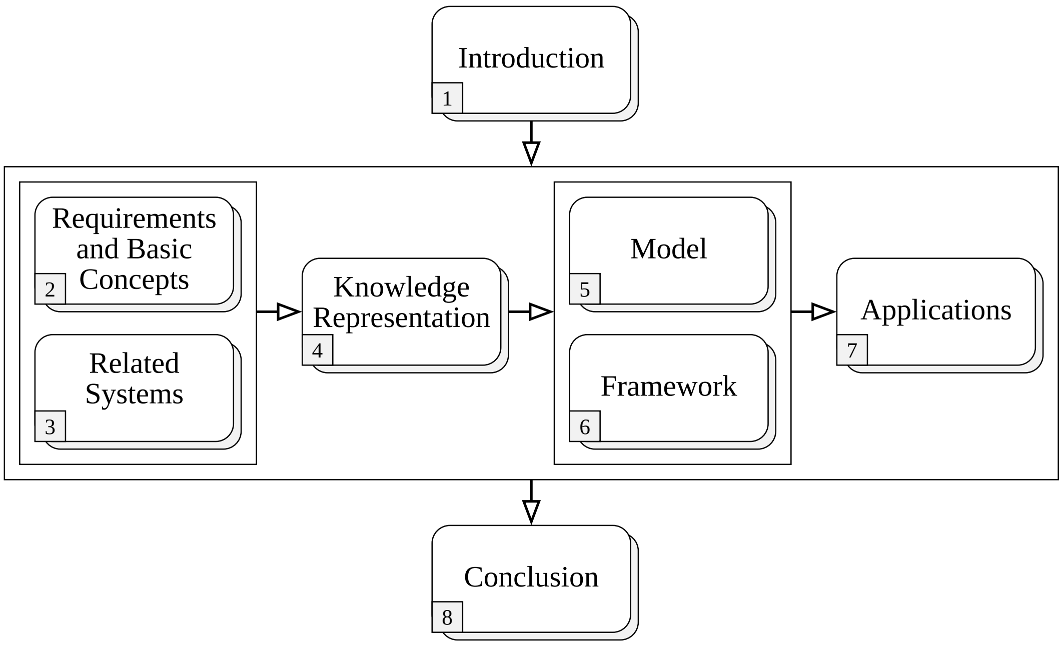


Figure 1.6: Outline of the thesis

they are given. We explain how the knowledge domain is structured, our choice of an ontological representation, the operations we use to access and manipulate the knowledge base, and related important concepts and methods.

5. In *A Model for Generating Multi-Party Conversational Behavior*, we present our conversational model that builds on the theoretical tools introduced in the chapter on related work. We explain the concept of individual dialogue engines for each conversation participant, and the model we developed to structure elements of discourse action in a three-level hierarchy of conversation activities, games, and acts. We also describe how the model allows encapsulated conversation participants to loosely collaborate on joint goals via the dialogue game metaphor.

6. *Realization of a Conversational Behavior Generation Framework* describes how a working, real-time capable, and flexible implementation of the conversational model can be constructed. The chapter begins with an overview of the multi-agent framework that integrates the CDEs and interfaces with the rest of the dialog system and then shows how activities are executed in it. Finally, it treats the integration of an external planner into the framework.

7. In the chapter *Applications Implemented Using the Behavior Generation Framework*, we show how the approach was used to implement action management in three dialogue systems, *VirtualHuman*, *OMDIP*, and the student project *Clue*, that have significantly different scope. Each system poses its own particular set of requirements. We show that the flexibility of the framework was sufficient to tackle them by describing the solutions in each case.

8. In the last chapter, *Conclusion*, we summarize and discuss the results of our work and evaluate how it meets our targets. Finally, we give an outlook of remaining issues and promising avenues for future work.

The appendices contain some additional references, including a description of the *Lisa* language that is used to specify action plans, the *JenaLite* API that was designed as a lightweight interface to ontological data structures, XML schemata for dialogue system definition (DSD) and directionML documents, as well as an overview of the system-independent dialogue branch of the ontology.

1.6 Notes to the Reader

- **Pronouns**

  With respect to the gender of pronouns we use the following convention inspired by (Hulstijn, 2000b): the speaker or initiator of a conversation is a “she”, the hearer or responder is a “he”. The user initiates the conversation and will thus be referred to as “she”. A system or virtual character is referred to by “it”, except for characters that have a name.

- **Definitions**

  For purposes of clarity, at some points we used concepts in definitions at a point in the thesis where they themselves are not yet (formally) defined. In these cases, and in cases where the definitions are not close together in the text, we provide an informal account for the meaning of these concepts in the surrounding description and give a reference to the section where the actual definition is given.

- **XML Shorthand**

  To encode and transmit data, our framework frequently uses XML structures, and several examples of such structures occur in the text. Because XML documents are quite verbose, and including them verbatim would take up considerable space, they are abbreviated or abridged in some places (however keeping the gist of the message). We also use a simple shorthand notation that cuts the required space approximately in half. The notation is described in Appendix E.

Chapter 2

Requirements and Basic Concepts

2.1 Introduction

A dialogue system is a computer system that can partake in communicative exchange in a natural language, and possibly additional modalities. Realistically, present-day dialogue systems can only cover a very limited subset of the full expressivity that is offered by human communication; to achieve full mastery in this area is generally considered to be an "AI-complete" problem, i. e., a problem that is likely to require an intelligence level comparable to a human's to be tackled. As long as the ability of computer systems to conduct a conversation is still a long way from approaching human competence, the practical rationale of using computers—to provide an effective and convenient tool to support humans—should, in our opinion, take precedence over attempts at modeling human discourse comprehensively.

Most current dialogue systems focus on restricted conversations that involve the human user and the computer collaborating to address some well-defined *task* that involves, for example, information seeking, controlling devices, or tutoring on some subject. (Allen et al., 2001a, p. 3) call interactions of this kind *practical dialogues*, and offers a *Practical Dialogue Hypothesis* that proposes

> *"The conversational competence required for practical dialogues, while still complex, is significantly simpler to achieve than general human conversational competence"*

We think that this is a plausible assumption, and that it is not restricted to purely task-oriented dialogues, but can also be applied to other related domains, such as simulated narratives, to a certain extent. The reason for this is that the hypothesis derives from the observation that focusing on a specific task narrows down the forms of interaction that are expected to play a role in the conversation, as well as the domain of relevant knowledge (namely, the knowledge directly related to and needed for accomplishing the task). A number of aspects of human interaction that are difficult to capture for computerized models, such as ironic or ambiguous statements, do seldom occur in predominantly task-oriented conversations[1]. More importantly, when these aspects are not essentially needed to accomplish a task, human

[1]Note, however, that there have been successful attempts to recognize irony in task-oriented dialogues, e. g., by exploiting cross-modal incongruities in the *SmartKom* system (Wahlster, 2006).

technique	**example task**	**dialogue phenomena handled**
finite state script	long-distance dialing	user answers questions
frame based	getting train arrival and departure information	user asks questions, simple clarifications by the system
set of contexts	travel booking agent	shifts between predetermined topics
plan based	kitchen design consultant	dynamically generated topic structures, collaborative negotiation dialogues
agent based	disaster relief management	different modalities (e. g. planned world and actual world)

Figure 2.1: Domain and interaction complexity for task-oriented dialogue systems with various approaches (from (Allen et al., 2001a, p. 2))

users usually do not expect nor demand from such a system that it is able to handle them. In this respect, typical human-computer interactions exhibit significantly different patterns than interactions between humans (Dahlbäck and Jönsson, 1992), which indicates that human users, to a degree, know about and accept such limitations, and consciously or unconsciously adapt their behavior.

A second relevant hypothesis, also from (Allen et al., 2001a, p. 3) is the *Domain-Independence Hypothesis*:

> *"Within the genre of practical dialogue, the bulk of the complexity in the language interpretation and dialogue management is independent of the task being performed"*

This hypothesis builds on the observation that practical dialogues share many interaction patterns, such as the form of question-answer exchanges, that occur universally and in the same way regardless of the actual task. If it is justified, it should be possible to address these general issues independently of any concrete system or application, and re-use patterns across applications.

In addition to task-oriented dialogues, we are also interested in another genre for dialogue systems, namely *interactive narratives*, like the *ZAMB* game. This kind of system allows the user to experience a story interactively. Like in the *ZAMB* scenario, such stories frequently include several characters—the user may or may not be one of them—that interact with each other by conversation or otherwise; this is in contrast to most task-oriented systems that lets one human user collaborate with the computer personified as a single agent. Because of this setup, interactive narratives offer a natural platform to study dialogical interaction involving more than two participants, i. e., multi-party interaction. Like task-oriented dialogues, most interactive narratives also feature only a restricted interaction: Besides the problem of comprehensively understanding all possible user interactions, the system designer usually needs to constrain them to ensure the progression of the story plot towards a goal outcome (or possibly several alternative ones), and the subject of the conversation is expected to relate to the elements of the story (see, e. g., (Riedl et al., 2003; Cavazza et al., 2002)).[2]

[2]The *Façade* system does not pose such restrictions, but simply ignores any user input it cannot understand.

Dialogue management is a very wide and multifaceted field of research. A comprehensive account of even just the major aspects would not be possible (or adequate) in the scope of this thesis. Also, what is required of a dialogue manager heavily depends on the particularities of the application(s) it is intended for. Figure 2.1 from (Allen et al., 2001a) shows a comparison of the power of different techniques being used for dialogue management, and the types of applications and phenomena that can be handled by instances of each paradigm. From top to bottom, the tasks increase in complexity and variability, and more complex paradigms can handle increasingly sophisticated dialogue phenomena. The additional reasoning power, however, comes at the price of increasing effort for modelling the domain as well as computational cost.

The next section examines the characteristics of the two types of systems we want to consider, "task-oriented systems" and "interactive narratives". After that, a section points out the role of the interaction environment and the perspectives of the user and the dialogue designer. Section 2.4 goes into more linguistic issues of modeling dialogues and conversations. Section 2.5 is about aspects related to dialogue management and covers patterns for communicative exchange, issues arising from multi-party interaction, and the modular architecture of dialogue systems.

2.2 Characterization of the Task

Two of the most important questions that must be answered to characterize and understand any interactive system are: *"What purpose is the system used for, in what context, and by whom?"* and *"What constitutes a successful interaction?"*—in other words, *"What must happen to achieve satisfaction for human users as well as the provider of the system?"*.[3] The answers to questions are different for the two system types we are concerned with: task-oriented dialogue systems and interactive narratives. We describe what characterizes both types of systems, and give some examples of what criteria can be applied to determine interaction success and failure in both cases. Afterwards, we take a brief look at related kinds of systems that represent special cases or a mixture of both types.

2.2.1 Task-Oriented Systems

The great majority of realized dialogue systems today are so-called *task-oriented* systems that can be used to complete a task, or to have a problem solved in interaction with the user. The dictionary meaning of "task" is *"a piece or amount of work set or undertaken"* (Chambers Editors, 1993), often in a given amount of time. Task-oriented systems often offer support for tasks by providing an interface to external applications to make their functionality accessible through dialogical interaction.

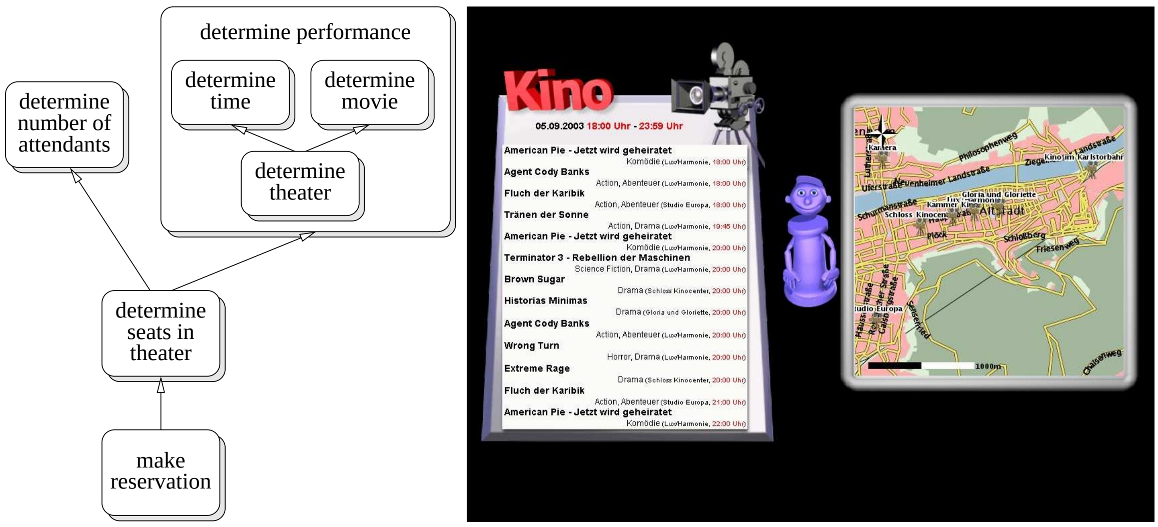
Examples of common tasks addressed by task-oriented systems involve information seeking, command-and-control interactions, collaborative problem solving, transactions such as ticket ordering, or customer help applications. *Assistive* systems provide help or guidance on some

[3]These criteria need not be identical; a system implementing on-line banking could easily satisfy users by making errors that benefit the customer, but this would not be in the interest of a financial institution that provided the system.

subject without themselves solving a task. Tutoring assistive systems often also make use of a narrative element. On the low end, speech-driven interfaces are used frequently today for the single purpose of navigation in menu-like structures, e. g. in phone applications. Service call centers use this technology to narrow down the possible reason for calling to guide the caller to a specialized human operator.

2.2.1.1 Task Structure

Focusing on task-oriented dialogue offers some advantages for the designer of a dialogue system. There are some common characteristics of task-oriented dialogues that do not extend to general dialogues.



05.09.2003 18:00 Uhr - 23:59 Uhr

American Pie - Jetzt wird geheiratet
Komödie (Lux/Harmonie, 18:00 Uhr)
Agent Cody Banks
Action, Abenteuer (Lux/Harmonie, 18:00 Uhr)
Fluch der Karibik
Action, Abenteuer (Studio Europa, 18:00 Uhr)
Tränen der Sonne
Action, Drama (Lux/Harmonie, 19:45 Uhr)
American Pie - Jetzt wird geheiratet
Komödie (Lux/Harmonie, 20:00 Uhr)
Terminator 3 - Rebellion der Maschinen
Science Fiction, Drama (Lux/Harmonie, 20:00 Uhr)
Brown Sugar
Drama (Schloss Kinocenter, 20:00 Uhr)
Historias Minimas
Drama (Gloria und Gloriette, 20:00 Uhr)
Agent Cody Banks
Action, Abenteuer (Lux/Harmonie, 20:00 Uhr)
Wrong Turn
Horror, Drama (Lux/Harmonie, 20:00 Uhr)
Extreme Rage
Drama (Schloss Kinocenter, 20:00 Uhr)
Fluch der Karibik
Action, Abenteuer (Studio Europa, 21:00 Uhr)
American Pie - Jetzt wird geheiratet
Komödie (Lux/Harmonie, 22:00 Uhr)

Figure 2.2: An example from a task-oriented cinema reservation application (*SmartKom*): the diagram shows the subtask dependencies (an arrow means that a step in the application is dependent upon the completion of another step)

- Many common tasks consist of successive steps or stages that depend on each other and have to be taken in some order, and it is relatively easy to specify constraints regarding which part of the task requires what preconditions, and to identify the corresponding dialogue segments. Also, sub-tasks can often be ordered hierarchically. Ideally, the different sub-tasks are self-contained, and their precise specification can change independently of other parts of the application.

  An example for this is an application for making cinema reservations with a structure as shown on the left side of Figure 2.2. The following is a corresponding dialogue from the *SmartKom* movie application:

(1)	USER:	*What movies are running tonight?*
(2)	SMARTAKUS:	[presents a list of movie performances (including performing theater) from 20pm the same day, and a city map where theaters are shown] *Here you see a list of performances running tonight. On the map, the cinemas are marked.*
(3)	USER:	*I'd like to make two reservations for this* [↗ selects performance] *performance.*
(4)	SMARTAKUS:	[shows cinema layout] *show me where you want to sit.*
(5)	USER:	*I want to sit* [↗ encircling gesture] *here.*
(6)	SMARTAKUS:	[marks two seats] *Is that correct?*
(7)	USER:	*Yes.*
(8)	SMARTAKUS:	*I reserved your seats. Your reservation number is 20. Please get your thickets no later than 30 minutes before the movie starts.*

The sub-tasks of specifying the performance to attend and the number of attendants do not depend on each other, but the actual selection of seats can only happen when both have been completed. Determining the performance is a sub-task that can be further broken up into selecting a movie and a preferred performance time, and then selecting from theaters where the movie is being shown at that time. The outcome of the sub-task could also be accomplished in a different way (e. g., the user could pick a performance directly from a list of possibilities), without affecting the rest of the task.

- For task-oriented systems, the domain can often be restricted to a small set of topics that will conceivably be addressed in the course of solving the task; the dialogue does not have to deal with issues beyond these topics. The set of objects occurring in the dialogue is determined by the task structure. A task-oriented system handling reservations for cinema performances will mainly have to deal with attributes of performance and movie objects, such as playing times, movie names, ticket prices, and so on. Although the concrete objects the dialogue is about may change over time (e. g., the movie selection is updated every week), these changes only affect content "parameters", and not the underlying structure of the task. A limited, well-defined, and closed domain makes it easier to construct the knowledge base for the system.

- The conditions at the start and end of the dialogue can be described in logical terms. Task-oriented dialogues can be segmented into stages with well-defined dependencies more readily than general free dialogue. The conditions for successful completion of the tasks are often relatively straightforward. In our example, the available performances and seats are known to the system, and it is clear which pieces of data have to be collected from the user to achieve a successful reservation.

- In many cases, the roles of the agents involved in the task are predetermined by the application and do not change during the interaction. This is also the case in the example, where the system takes the role of the vendor and provider of possible performance configurations, and the user is the customer who provides missing data and selects from the available choices.

These points indicate that task-oriented dialogue is an easier terrain for research than common every-day communication, because some trickier phenomena, like off-topic dialogue, fuzzy and changing objectives, or unclear situations are less likely to be found, and also less relevant to the task's completion.

2.2.1.2 Interaction Structure

In many cases, the task can already help to structure the dialogue in stages and dependencies. It also constrains the types of utterances that are to be expected because they are sensible or mandatory at different points in the task-solving process. The vocabulary is also limited, since the situation of solving a task provides for a rational setting. As mentioned in the introduction of this chapter, many users will tend to adapt to this and use plain language with simple and concise utterances, and also appreciate it when the system acts likewise. In the extreme, such a dialogue can be reduced to "command-and-control" language. If the user has some knowledge about what is involved in the task, she will also have an a priori idea of what she can expect to talk about, and will be motivated to stay within the limits of the task-related language, because she is primarily interested in accomplishing the task.

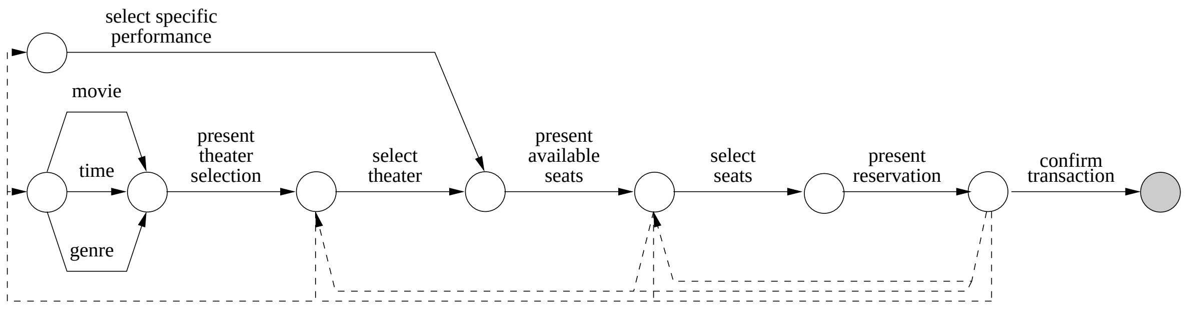


Figure 2.3: Example structure of a cinema reservation dialogue

Consider again the task of making a cinema reservation. The user and the system need to agree about a specific instance of a reservation, which is characterized by the time of the performance, the name of the movie, and few other possible parameters. In this case, the bulk of the dialogue will consist of telling and asking for values of attributes of the domain objects, and agreeing on a configuration of values for the final reservation transaction. Figure 2.3 shows a diagram of the possible interaction course in such an application. It closely follows the dependency structure of the task (cf. figure 2.2). It allows transitions back to earlier stages of the interaction to change earlier commitments, e. g., if the user is not content with the available seats for a selected performance.

Some basic patterns are reoccurring in task-oriented interaction. The narrowing down of a concrete instance from a set of alternatives can often be modelled by a sequence of *form-filling* interactions. A form-filling interaction is concerned with the concretion of one or more underspecified objects, like forms where empty slots have to be filled in. The sample interaction has the final goal to agree on one fully specified, unique object representing the desired reservation by subsequently determining the concrete values for the unfilled slots, thus reducing the set of alternatives. Besides form-filling actions, *requests for information* and *providing* *information* (either in answer to a request or as an initiative) are important interaction types in task-oriented systems. In the example, the user may request information about the value space given a partial instantiation, e. g., *"What action films are there on Saturday?"*. Information requests can also be posed by system initiative, e. g., *"What genre are you interested in?"*. The kind of information that is exchanged in these interactions depends on the application. In a cooperative task, a participant could, e. g., also request another to propose a next step in a cooperative plan towards the goal, or to execute such a step. If explicit *commands* are issued, they are mostly directed from the human user to the system. Finally, some meta-interactions are also frequent. *Dialogue control interactions* serve to reach common ground. This occurs on the level of dialogue acts (e. g., backchannels, nodded agreement) or on the discourse level, e. g., to confirm explicit agreement before undertaking an action that is not reversable (such as the confirmation of the reservation parameters before executing the reservation in the example). Some additional types of (meta-)interaction might address providing help with the task, recovering from errors and misunderstandings, or conventional interactions for greetings and ending the conversation.

To summarize, the general interaction structure in task-oriented dialogues is directed towards the completion of the task. The participants tend to subsequently *reduce the number of alternatives* for task parameters, and take steps *forward towards the task goal* as a logical end point.

2.2.1.3 Purpose and Success Criteria

The foremost purpose of a task-oriented system is to allow the user to complete one or more associated tasks or transactions. Completion in this case can mean different things. For example, an interaction with a system that offers the opportunity to get information about cinema programmes and possibly also to purchase tickets can be deemed to be successful even when no ticket is sold, if the reason for this is that there is no performance on offer the user is interested in. Additional important requirements include robustness of the interaction, i. e., recovering from errors either in the user interaction or in the associated application. This is important in applications that employ speech or multimodal input, since in addition to user errors, they also have to deal with possible misrecognitions of actually correct user input. The interaction should be efficient, i. e., the user should not be required to make redundant turns. In many cases, the system should be usable by untrained users, which is again a special issue for spoken dialogue systems, since they can only sport a restricted vocabulary. Whether or not, and how successfully, a task was completed lends itself to quantitative evaluation. It is often straightforward to count the percentage of successful tries, count the number of steps that were taken, or identify misleading steps.

There are approaches to a formal evaluation of dialogue system usability for task-oriented systems, such as the *PARADISE* framework (Walker et al., 1997). The general structure of *PARADISE*'s objectives for measuring spoken dialogue performance is shown in figure 2.4. The overall goal is to maximize the user's satisfaction. This breaks down into task success on one side, captured using the *kappa coefficient* (Carletta, 1996), and the costs for the user which are in turn further divided into efficiency measures (such as number of required turns and overall time taken for the task) and qualitative measures (such as how much delay system responses take, and how many errors occur during the interaction). For a given system, the

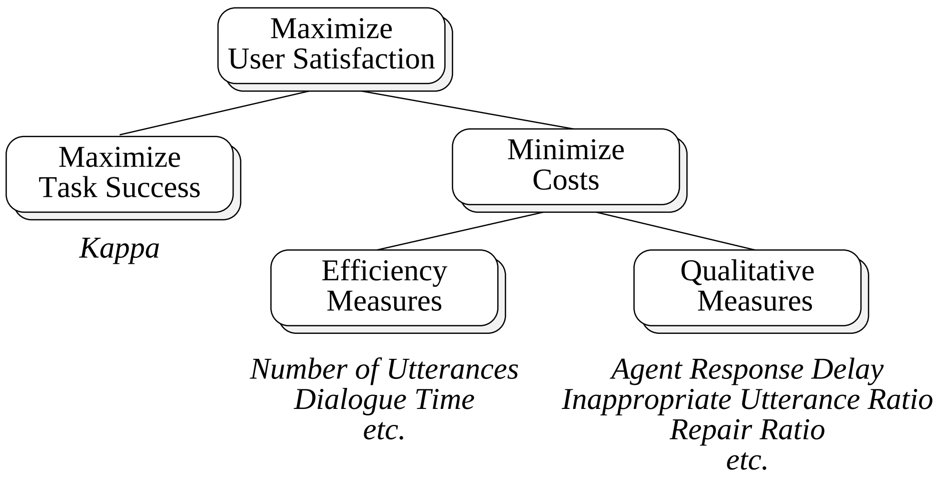


Figure 2.4: *PARADISE* objective structure

success in meeting the criteria can be measured by letting subjects interact with the system and evaluating the performance afterwards. Part of the data can be collected automatically by logging interaction data and transcripts (e. g., elapsed time, task completion, number of timeouts). Some more subjective measures are taken by letting the subject complete questionnaires after the interaction with questions such as *"Was the system easy to understand in this conversation?"* or *"Did the system work the way you expected it to in this conversation?"* (see (Walker et al., 2000) for a comparative evaluation of the *ELVIS*, *ANNIE* and *TOOT* systems and further methodological details). (Schiel, 2006) points out that difficulties may arise when *PARADISE* is applied to more complex task-oriented systems, especially multimodal ones, citing the example of the *SmartKom* system. He cites as most problematic that (a) correctness labels cannot always be assigned to single recognition results due to the multiple asynchronous modalities, and (b) the task structure is much more fuzzy than simple database access interactions, and there may be more than one way to solve a task. For such systems, an adaptation of *PARADISE* called *PROMISE* is proposed (Beringer et al., 2002).

A more comprehensive evaluation methodology is provided by the *Voice Application Performance Index* (VAPI) used in the *Voice Award*. The Voice Award is a prize that has been awarded annually since 2004 for the best German-language speech applications deployed for business use. It involves four benchmark criteria (Hoffmeister et al., 2007):

- **Technical Performance Characteristics**

  This includes the number of possible simultaneous user connections, the number of calls per day, the number of unique users, the kind of speech recognizer, and the number of interfaces to other systems.

- **Cost Effectiveness**

  This criterion includes the costs per call, the average task completion rate, how long it took to build and deploy the system, and the time it takes to amortize the costs of the system.

- **Voice User Interface**

  Sub-factors for the user interface are understandability, effectiveness of use, recognition errors and their impact on the dialogue, and "hear and feel" in terms of sympathy and appropriateness for the task.

- **Innovation**

  This criterion accounts for new functionalities, innovative user interfaces allowing a simpler or faster use, or novel business models

The results across all criteria are integrated in a "VAPI scorecard" model where the ratings for the different criteria are weighted according to expert assessment with the aim to ensure a balance of usability and technical/business factors. The test is designed to provide a unified, structured benchmark methodology for applications of different purposes.

2.2.2 Interactive Narratives

Another emerging purpose for dialogue systems is to provide entertainment by telling or enacting a story. We call such systems *narrative* systems, or, if the user can participate in the narrative, *interactive narratives*. In the entertainment and infotainment industries, there is an increasing interest in telling stories via the computer that allow for active participation of a human user that communicates with virtual characters and/or a virtual story world. A storytelling system focusing on entertainment or edutainment is successful when its audience is presented with a convincing and immersive experience; in addition to bringing the point across, it must, above all, not be boring.

One can think of interactive narratives as simulated theater performances that are enacted by virtual characters that can play assigned roles and that allow for user participation. Another perspective is to view them as a kind of computer game, and in a limited sense, computer games such as *Deus Ex* often use interactive scenes where the player does interact with the game characters to influence the course of the subsequent game narrative. However, these scenes are usually quite inflexible in that they allow the player to select between a small set of alternative options of what to say, and the resulting space of outcomes is easily enumerated (and consequently just realized as a tree of alternatives). In such situations, the gameplay proper is suspended and resumes after the interaction has been completed. This has the effect that (a) the narrative aspect is detached from the actual gaming experience, actually just an "add-on" to provide atmosphere, and (b) there is no real sense of agency for the player.

2.2.2.1 Task Structure

Interactive narratives are mainly driven by two forces: the intention (provided by the dialogue designer and enacted by the system) to tell a story, and that of the user to experience, influence and drive forward the story forward through own actions. Like in a drama, longer interactions are often segmented into successive units, or *scenes*, that might involve different characters and environments, and whose order is determined by dramatic intent rather than logical necessity.

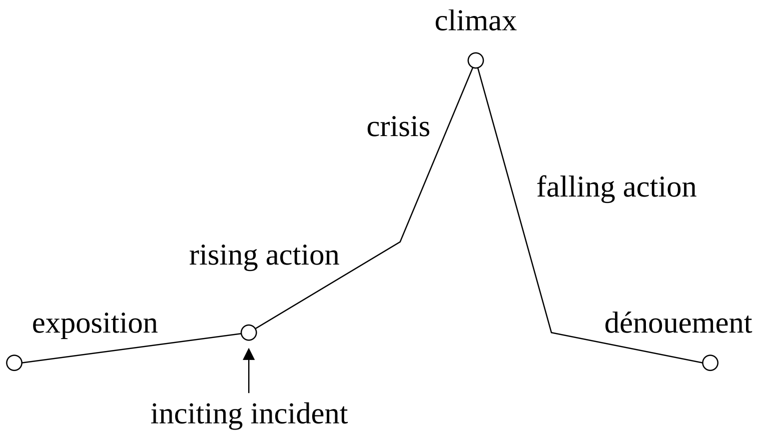


Figure 2.5: The Aristotelian dramatic arc (from (Mateas, 2002))

If an interactive narrative follows a classic arc of dramatic tension like in an Aristotelian drama (see figure 2.5), the goal of the interaction is not the conclusion ("dénouement") at the end of the narrative, but the climax. It is the intention of the dialogue designer to let the events move towards it at a pace that also allows the story to unfold. In contrast to task-oriented interactions, the focus is not to find a solution to a task quickly and efficiently, but to *demonstrate* the consequences the actions of the user entail, and concluding the story with the consequences. The protagonists of the story, however, whether their roles are taken by human users or virtual characters, *do* have tasks to do and goals to achieve in a story.

There are several methods for story control in narrative systems. The *emergent narrative* paradigm (Aylett, 1999; Aylett et al., 2006) aims at designing fully autonomous characters that are driven to enact the story from inherent desires and their perceptions of the environment. This approach allowing the characters *strong autonomy* is criticized by, e. g., (Mateas and Stern, 2000), on the grounds that the selection of "correct" actions in the context of a desired story cannot always be made on the (local) basis of the individual character's knowledge alone; they argue that there must be a supervising entity that enforces the intended story from a global omniscient perspective. This direction is taken by systems that employ *drama managers* which assume full control of the behavior of all characters, possibly with the exception of secondary behavior that is not relevant to the story (such as idle animations). The drama manager can, for example, restrict the outcomes of the story by a series of branching points that create a tree of possible stories, a technique that is frequently used in computer games (Lindley, 2005); branching story structures are, however, hard to manage for more complex narratives or ones that allow the user extensive freedom of action. Another possibility is to let the drama manager observe the events in the environment and dynamically adapt the story. In some situations it can also be necessary for a drama manager to prevent user interactions that threaten the story, or to trigger interventions, e. g., to manipulate the outcomes of interactions, or to create events that serve to repair the storyline. This technique is used, e. g., by the *story mediation* approach proposed by (Riedl et al., 2003).

This thesis is not directly concerned with the problem of drama management. However, for systems that have a component that acts as a narrative control instance—such as *VirtualHuman*'s narration engine—our model offers the possibility to dynamically balance control and autonomy (Löckelt et al., 2005). It puts the virtual characters in the role of actors that autonomously perform actions that achieve goals set by directions from the control instance. To enable the drama manager to guide the story, and intervene in situations where the outcome of the story is threatened, it also provides the possibility to deliver feedback about the current state of the interaction. The drama manager can use this information to adapt future directions, or to retract goals and replace them with new ones.

For interactive narratives, an element of chance in the task structure is much more acceptable than for task-oriented systems, or can also be explicitly intended to create variation. If a virtual character tells the same story twice in a row, possibly even using the same utterances, the entertainment value as well as the believability of the agent as a "living and thinking entity" will suffer. Users may forgive small incoherencies (e. g., caused by overzealous or "emergency" mediation) if the story is interesting otherwise. Robustness of understanding typically is not as crucial as in task-oriented systems, if there is no time limit or important task involved. On the other hand, poor understanding and presentation can also negatively affect the aesthetic experience and the immersion of the user.

2.2.2.2 Interaction Structure

Like task-oriented interactions, interactive narratives exhibit reoccurring building blocks on several levels. For one, general dialogue patterns like question/answer exchanges, etc. also apply in this context. However, there are also building blocks on higher levels, as outlined by, e. g., (Propp, 1968). In his investigations on the morphology of the folk tale, Propp identifies a set of archetypical dramatic structures present in virtually all classic narratives. His findings are also applicable to more contemporary works, as has been shown for, e. g., the *Star Wars* series (Hiltunen, 2002). The structures include archetypes or roles for the dramatis personae (hero, villain, a "helper" figure, etc.), motivations (such as "rescue the princess"), typical activities and scenes (e. g., the exposition, or an "end-fight" climax with a villainous character) and instruments for the story.

In narratives, longer stories are usually split up into several *scenes*, where each scene communicates a certain purpose, goal, or *theme*, in the story. Between scenes, the setting of the narrative, e. g., the virtual physical environment, or the set of participating characters, can change. They can be seen to correspond to sub-tasks in task-oriented interactions. A scene can be further subdivided into smaller parts. For this, a widespread concept in interactive narratives is the notion of the story proceeding in *dramatic beats*. Mateas and Stern describe a (dramatic) beat in the following way:

> *"First, beats are the smallest unit of dramatic value change. They are the fundamental building blocks of the interactive story. Second, beats are the fundamental unit of character guidance. The beat defines the granularity of plot/character interaction. Finally, the beat is the fundamental unit of player interaction. The beat is the smallest granularity at which the player can engage in meaningful (having meaning for the story) interaction."* (Mateas and Stern, 2000, p. 4)

A dramatic beat does not necessarily involve only singular actions. For example, in the *Façade* system described in the next chapter, dramatic beats are shared between several involved characters and represent joint plans for action.

In comparison with purely task-oriented interactions, the dialogue designer will want to use more variation of expression, because of the underlying goal to entertain people. Besides more elaborate language, multimodal presentations are useful to provide variation because there can be more than one way to present an event, as well as to broaden the range of expression the characters are capable of in a single action. Modalities other than voice can often be used to convey a character's emotions in a more believable, direct, and concise manner. Users will also have a greater tendency to experiment with the system to explore the boundaries of what it can handle. In interactive narratives, it can also add to the atmosphere to consider how the status of the participants and their social relations affect character behaviors, e. g., by influencing the participation rate in the conversation (cf. (Rumpler, 2007) and Section 2.5.2.2 on participant roles in multi-party conversations).

2.2.2.3 Purpose and Success Criteria

The main purpose of the characters of a narrative system (corresponding to the protagonists and supporting roles) is different from the one in task-oriented systems. The characters should appear and act *life-like*, but this is not to be mistaken to mean that they have to be graphically photo-realistic, and neither do they have to exhibit near human-level intelligence or dialogue competence. Rather, they are supposed to exhibit life-like behavior *within the context that they are artificial entities* and not real persons. Animated characters like *Donald Duck* are not meant to appear human-like; but fortunately, this is not required for them to be convincing. It can even be counter-productive when realism is approached, but not quite achieved, a situation which was dubbed the "Uncanny Valley" by (Mori, 1970), which postulates that characters are perceived to be *more* unrealistic, or even spooky, when they are very close to realism without completely achieving it. A human interacting with a computer system may know fully well that computer characters do not really feel emotions, but this does not have to diminish the entertainment value of them expressing *simulated* emotional behavior. After all, this is exactly what human actors do when they perform. Virtual characters are successful if they manage to show some kind of *personality* that fits their role in the setting and is consistent over time (Gebhard, 2005).

Another point that was brought up in (Doyle, 2002) is that the main challenge for traditional rational agents, and indeed agents in task-oriented systems, is the selection of actions necessary to reach some goal. In other words, it is required that they be *effective*. Storytelling protagonists that are meant to be life-like, in contrast, predominantly have to be *believable* and *interesting*, attributes that depend heavily on expression of action as well as emotion. This means that, e. g., acting in an identical manner in identical circumstances, while being perfectly rational, might not constitute desirable behavior for a life-like character, since determinism is perceived as odd for a sentient being. Acting irrationally, on the other hand, can be just fine for the character as long as it is plausible. In contrast to task-oriented systems, inconsistency or lack of transparency can be tolerable (e. g., in order to not give away the story in advance) or even necessary (e. g., to save the storyline) in interactive narratives.

To sustain the flow of the narrative, it is even more crucial than in task-oriented systems to have fast (ideally real-time) reaction times. Generally, people do not even know how long it would take a human to book cinema tickets, or are prepared to rationalize and accept that "the agent needs to wait for the answer from the ticket database". However, since the "suspension of disbelief" responsible for an immersive experience must be upheld, un-natural delays during conversations are dangerous in an interactive narrative: they endanger the impression of interacting with a real, believable character.

The evaluation of an interactive narrative system relies more on subjective measures than with task-oriented systems. It is generally possible and meaningful to determine (analogous to task-oriented systems) whether some crucial points, or sub-tasks, of the story were realized, and whether the story was consistent. The overall success of an interaction, however, in this case does rely more on the qualitative experience of the user than on quantitative criteria.[4] Parameters that are important when a task should be accomplished effectively, like the number of utterances required, do play a lesser, or no, role.

2.2.3 Related Types of Systems

Systems that try to educate or instruct users about some subject form a hybrid of task-oriented and entertainment purposes. They need to incorporate a concept of didactic structure for their task, and also often involve a narrative component to make the interaction more interesting. An advantage of tutoring systems that use a virtual environment is the possibility to simulate situations that would be difficult to create in a real environment, dangerous to the trainee, would endanger valuable equipment, or all of the above. In comparison to narrative systems intended for entertainment, tutoring and help systems will tend to follow a more rigid storyline. Virtual tutoring systems are on the rise in military training. An elaborate example of a multimodal military training system we will examine in the next chapter is the Mission Rehearsal Exercise (*MRE*), which lets the user experience a conflict between mission objectives and handling a road accident. Other systems have been developed to train soldiers to deal with foreign cultural contexts (Deaton et al., 2005), to act as museum guides (Kopp et al., 2005) or to educate school children to cope with bullying situations (Paiva et al., 2004).

A special case, albeit a frequent one, is systems developed primarily for *research* purposes, e. g., in the fields of computational linguistics or human-computer interaction. These are often of the task-oriented type, which has been investigated most thoroughly. The purpose of research systems is to provide new insights by, e. g., demonstrating or explaining particular phenomena in dialogue, or to test a theory. Interactive systems with virtual characters are also created to test scientific hypotheses or to gather empirical data about human-human or human-computer interaction. Variants here include systems that conduct a full dialogue with users or transcribe recorded or live interactions they do not take part in, e. g., the *AMI/AMIDA* projects.[5] There also are setups that are controlled by a human operator to simulate dialogue systems that do not actually exist in so-called "Wizard-of-Oz" experiments; this technique can be used to gather information about an application scenario before or during the development of actual prototypes (see, e. g., (Schiel, 2006)).

With increasing realism, virtual characters can also to some degree take the part of human counterparts in research settings. A recent interesting example is a piece of psychological research reported by (Slater et al., 2006): The well-known *Milgram Experiment* (Milgram, 1963) that had raised considerable concerns because it required the test subjects to treat humans in an unethical way, could be repeated with virtual characters and yielded results

[4]cf. the *VirtualHuman* evaluation with school children reported in (Langer et al., 2005)

[5]Project website: *http://www.amiproject.org*

that are comparable to the original setup. In such cases, the system is designed to capture one or several phenomena occurring in dialogues, or to measure a psychological response from the user. A research system meets its purpose successfully if it provides meaningful data on the adequateness of the scientific models it uses. In research systems, the system designers are often identical with the users.

2.2.4 Summary

We examine conversations that are driven by the goal to complete a given predefined task, or to realize a storyline, respectively. Both conversation types have in common that during the interaction, a set of preconditions is established that are necessary to arrive at the goal. In both cases, we assume that the necessary "plan" is *not* negotiated between the participants—that is, we explicitly do not include the particularities of "negotiation dialogues", which would introduce a whole additional field of research (see, e. g., (Hulstijn, 2000b) for a comprehensive treatment of negotiation dialogues).

There are some major differences between task-oriented interactions and interactive narratives. The former are intended to be efficient, while the latter should be entertaining, which means that their success cannot be measured quantitatively by "number of turns until completion". Task-oriented interactions are mainly determined by the task description, and what the application can and cannot do, while interactive narratives have additional story constraints. Thirdly, the task structure differs significantly as task-oriented interactions have a tendency to move towards a solution; in a narrative, while there is also an ultimate goal, the whole point of a story is to throw in some complications and perplexities in between to create dramatic tension.

2.3 The Interaction Triangle

In this section, we examine the relationship between human users and virtual characters, the system itself, and the dialogue designer. This relationship can take on several forms depending on the number of users and virtual characters, and the interaction possibilities between them.

Figure 2.6 shows a succession of conversational system constellations of human users and virtual characters from a purely system-initiated monologue over several steps with increasing interactivity and number of participants up to applications where a variable number of human and virtual participants interact. It also reflects a progression of stages in the development of dialogue systems towards increased complexity. There also are systems that deal with purely human-human conversations, e. g. the *AMI/AMIDA* projects, whose goal is to record and summarize meetings; but we do not consider this configuration here.

A *system monologue* is the simplest, unidirectional form of a conversational system. An *interactive* system where both the user and the system can contribute has to deal with a whole new set of issues. A *simulated conversation* involves computer characters interacting with each other, while the interaction can be observed by the human user. One step further are

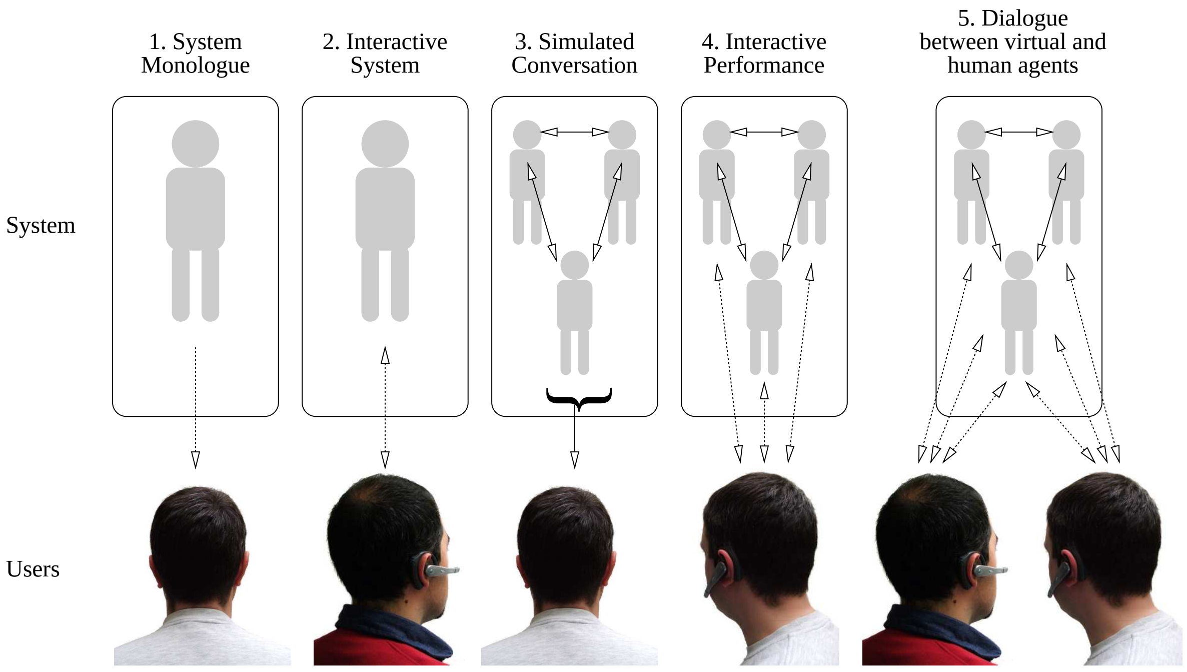

1. System Monologue
2. Interactive System
3. Simulated Conversation
4. Interactive Performance
5. Dialogue between virtual and human agents
System
Users

Figure 2.6: Complexity progression of conversational systems (from (Wahlster, 2005))

interactive performances, where the user can also participate in the interaction. The final scenario type, of which the *VirtualHuman* system is an instance, also involves multiple users, and represents the focus and state of the art of current research.

An additional dimension that has gained importance since real-world applications of growing complexity are tackled by dialogue systems is the role of the *dialogue designer*. The dialogue designer has to provide the definition of the interaction environment, as well as the characters in terms of their knowledge and behavior. The goal of the designer is to arbitrate between the system's purpose and the needs and expectations of the users and to ensure that the system can conduct an interaction that is successful with respect to the application type. Figure 2.7 shows a triangle diagram illustrating how the users, the virtual characters, and the dialogue designer are related to the dialogue system's definition and operation. The system interconnects a number of human users and virtual characters as conversation participants. They partake in conversational (and possibly physical) *interactions* in the context of an interaction *environment*. The virtual characters and the environment together form the *dialogue system*. All participants also need to take into account all other participants and their actions in order to be able to *coordinate* their efforts at having a coherent interaction.

We first look at the interaction environment. Then we examine the content of interactions and the mode of the exchange of communicative action. We then in turn take the perspectives of the users, the virtual characters managed by the system, and the interaction designer.

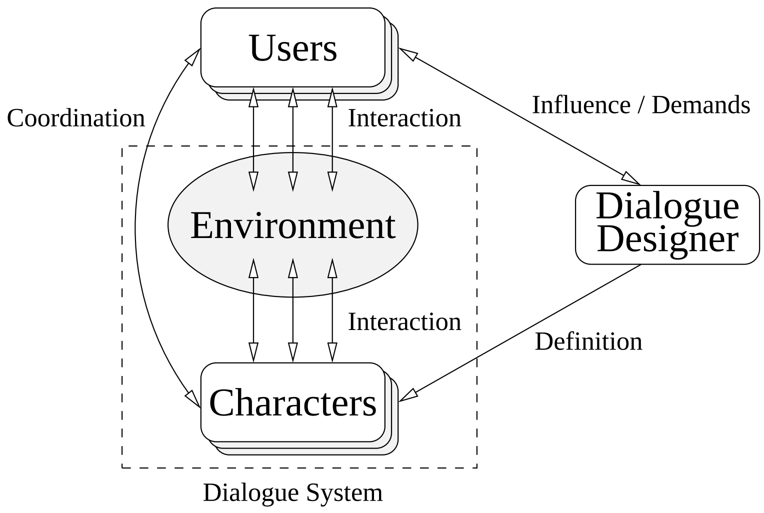


Figure 2.7: The Interaction Triangle

2.3.1 The Environment of the Interaction

We call the space in which the interaction takes place the *environment* of a dialogue system. In the real world, conversation always occurs in some kind of physical and mental environment. This environment must be taken into consideration, as it puts constraints on what can be done. Even for conversations over the telephone, where the participants do not share the same location, the physical environment has crucial impact, as anyone who has tried to call somebody from a crowded subway car can confirm. Human-Computer interactions often include some additional *virtual* or simulated physical environment by means of graphics or other modalities. One issue is whether or not the human conversation partner(s) are represented via avatars, i. e., actually *co-present* with the characters, in the environment. Co-presence enables a more immersive and direct interaction, but sharing space with virtual characters can also be perceived as a nuisance (as known e. g. from negative reactions of users to helper assistants in office applications that tend to get in the way of what the user intends to do).

If the conversation participants are embedded in a virtual environment, it is possible to have (virtual) physical actions in this environment. Virtual humans can use facial expressions, gestures, and other means of non-verbal expression. They can also manipulate simulated physical objects in this environment. They are agents that are *situated* in a virtual reality. The environment is, however, not restricted to the virtual physical setting, but also includes the wider *context* the system is located in. The real physical environment for the interaction with a dialogue system can consist of purely textual output showing written responses to user input entered via a keyboard, or it can be a fully-fledged 3D rendition. It is characterized by what it contains, how it is presented, and which interactions are possible with it for the conversation participants. These questions must be addressed in the knowledge representation of the system to solve the problem of *world representation*. Since a virtual environment represents a form of *space* to move in, the world representation must also include ways of modeling the possible spatial configurations and relationships.

For many applications, it is adequate and beneficial to restrict the environment to components that are immediately relevant to the task at hand. It can be confusing to add further elements, for example for decoration. In human conversations in the real world, objects of the physical environment may be off-topic, but they are never really off-limits to be referred in the conversation. The same holds, in principle, for conversation topics. However, as stated in the chapter's introduction, human users are generally well aware that today's computer-generated virtual humans are not prepared to talk about just any topic. They also usually do not expect to be able to refer to all objects or parts of objects presented on-screen, for example, that a character represented by an avatar can talk about the color of the avatar's hair. In this case, it is beneficial to have an understanding of the user's intuition, and to try to guide it correctly instead of leading it astray. To achieve this, it is necessary to make reasonably clear which objects are "live" in the environment, and which are not, as well as to avoid unnecessary objects in the environment that cannot be referred to and that serve no purpose but decoration (a technique that is sometimes used is to visibly highlight objects that can be interacted with). This will help to prevent unnecessary user confusion and possibly frustration, since with respect to the interaction, an object is really *present* in the environment if, and only if, there are ways to refer to it and get a reaction.

The virtual environment can be just a passive display that is observable by the human user, or it can be connected to the real environment. If there is a one-to-one *coupling* between the virtual and the real environment such that manipulations on a real object change a virtual object and vice versa, the setting is called *mixed reality* environment. (Kruppa et al., 2005) describe such a setup in which a character "lives" in a room, can move about, and offers situated assistance to users within the environment. Virtual characters can also use effectors to cause changes on objects in the real world. The user, too, does not necessarily have to be restricted to, e. g., navigating a user avatar and giving conversation input. In the *COHIBIT* system (Kipp et al., 2006), the user can re-arrange real-world car model parts. The system senses their current positions (via RFID technology); with this information, the system's virtual characters can provide hints for the creation of a car design. Another interesting example of this is given in (Paiva, 2005): In the described storytelling system, a child can transmit its own emotional state to game characters by interacting with a sensor-equipped doll that reports, e. g., being shaken or cuddled. The doll acts as a proxy object connected to a virtual object placed in the real world, and is the origin of the game character's affective state.

2.3.2 User Perspective

2.3.2.1 Natural vs. Asymmetrical Interaction

In the general case, the human user will compare the conversational interaction with the interactions with fellow humans. She will value naturalness and ease of use. This is a challenge because all contemporary dialogue systems lack the knowledge as well as the intelligence to converse on a level on par with a human. The fundamental asymmetry in human-computer interaction is often overlooked. This includes both the user and the system itself, and in both cases involves generation as well as understanding.

Human users frequently experience problems when they are forced to restrict themselves to interactions the computer can process, or when they should find out what these interactions are in the first place. To this end, it has been proposed (Jönsson and Dahlbäck, 1988; Jönsson, 1997) that a dialogue system should exhibit *habitability*, e. g., by employing a consistent sub-language that can be easily grasped by the user and enables her to communicate fluently with minimal risk of straying outside the borders of understanding. Habitability can also be enhanced by other means, such as giving cues about expected input to the user. Generally, the system should

> *"... clearly show the user which actions it is able to perform, which initiatives it can respond to, which it cannot respond to, and why this is the case."* (Jönsson, 1993, p. 2).

How can habitability be achieved without extensively training the users? One possibility is to re-introduce symmetry by ensuring that the system only produces contributions that it would be able to understand if they came from the user, i. e., all generated acts must also be analyzable.[6] A drawback of such an approach is that this means that the system might have to do without more elaborate code, such as ambiguous idioms or figurative expressions, since they are harder to understand and may also encourage the user to use more complicated language. In some cases, especially in the case of an interactive narrative, this might hurt the atmosphere, because the language would be too plain and "matter-of-fact".

Habitability is also increased if the system is able to deal gracefully with errors (application failures as well as understanding problems), offers help for inexperienced users, or has the ability to dynamically adapt to different users. Additionally, systems can to a certain degree exploit the user's imagination by adhering to *principle of least surprise:* Even when the system does not achieve human competence in conversation, it should be possible for a human to predict which kind of utterances will probably be understood by the system. This is supported by consistency in the user interface, e. g., making available similar interaction possibilities in similar situations. If the dialogue system fails to understand the human, it should provide feedback about *why* this is the case (for example, missing vocabulary, or its limited capacity to understand the domain), to enable the human user to adapt to the level of the machine.

Taking the scenario into account, it is also an option to accept the fundamental interaction asymmetry without reservations. It would be counter-productive to outfit a system designed for a narrow domain, e. g., ticket sale, with a general-purpose lexicon. The inclusion of superfluous vocabulary that is unlikely to be used at all for the task of the system would negatively impact the recognition rate and introduce unnecessary ambiguities, hurting the effectiveness of the interaction. In such a case, interaction robustness must be weighed against the ability of the user to also address off-task topics.

2.3.2.2 Response Delays and Feedback

An interaction will be perceived as unnatural, and possibly inconvenient, if the time the system needs to respond is substantially longer than in typical human-human interactions. This problem can be ameliorated somewhat if there is at least some feedback from the system that

[6] One technique to help this would be adopt an approach of "no generation without *interpretation*" (analogous to "no generation without representation" (Wahlster, 2002)) – i. e., the system should only generate output that it could also interpret as input (cf. the concept of an "anticipation feedback loop" in (Ndiaye and Jameson, 1996))

acknowledges that an utterance has been "heard" and is being processed. Three important limits can be used as a rule of thumb to estimate how the interaction flow is perceived by the user (Miller, 1968):

- A response time of up to *0.1 second* is perceived as instantaneous reaction,
- *1.0 second* is noticed as a delay, but preserves the feeling of free interaction,
- Delays should stay below *10 seconds* to keep the user focused on the dialogue.

The acceptable response time for a given system is further dependent on the application and purpose. In a narrative context, long pauses will not just delay answers, but also disrupt the flow of the narrative. In the presence of pauses, it helps if the user can rationalize why "it takes a little longer", e. g., when the system explains that it has to connect to a database to answer some query before making a pause. In all cases, it is advisable that the system acknowledges user input and consistently signals in some way that it is still operational and/or occupied (e. g., by displaying idle or busy gestures).

2.3.2.3 Believability and Immersion

The acceptance of a system is strongly dependent on coherent behavior. Especially if virtual characters are involved, or the system purports to have a "personality", it must act *believably*. This does not necessarily have to mean that a character's actions have to be logically consistent, or human-like. The main requirement is that the character's behavior be plausible given the information (or mental image) the user has about its inner life. This can lead to situations where a character that is less human-like is actually more believable when exhibiting a certain behavior. It is frustrating to encounter a dialogue system that uses elaborate canned text, which leads to the expectation that it actually can engage in free conversation, only to earn repeated misunderstandings when it turns out that the understanding is not up to par with generation. A sensible user does not really assume that she is interacting with an agent that is intelligent on a human level. It is important, however, that she can construct a mental model that enables her to have some notion of what she can expect the agent to do.

To achieve believability in a narrative context, it is crucial to *express* actions in addition to the task of *selecting* them. For behavior to be believable, it must be *interesting* instead of *effective* (Doyle, 2002). An additional issue contributing to believability is the *variety* of utterances. Like what was said about interaction asymmetry, it can be acceptable to use a uniform utterance schema across cases that are similar. This is especially true for task-oriented systems. It can even be beneficial in this case to strive for a simplified, predictable, and unambiguous interaction, since the user can concentrate on the task and is not distracted. On the other hand, in systems that are predominantly meant to entertain, poor variation and overly plain language harms the atmosphere and believability. The emphasis with such systems is not on conveying information in the most efficient and reliable way.

Depending on the purpose of a system, *user immersion* has varying degrees of importance from crucial for acceptance to irrelevant. Part of the requirements for immersion is that the user experiences a sense of agency, that is, the actions of the user have a manifest effect on the environment. Narrative systems with life-like virtual characters also rely on the *Eliza Effect*, i. e., that the user implicitly assumes their actions to be guided by motives and thinking processes similar to a human's. The contributions of the dialogue partner(s) must be adequate (in the sense of (Grice, 1975)) in the conversation, i. e., exhibit the right degree of informativeness, be relevant and clear, and be founded on an understanding by the system. However, narrative systems that try to produce adequate feedback for atmospheric purposes, even on user input that was not understood properly, are in danger of failing miserably, like in the following example of a typical interaction with the *Eliza* system:

PATIENT:	*Like I said earlier, I hate the rain.*
ELIZA:	*Why do you like you said earlier, you hate the rain?*
PATIENT:	*What do you mean?*
ELIZA:	*Please elaborate on that.*

After just a short interaction, such behavior results in the immersion-shattering realization that the "understanding" exhibited by this system is really an illusion. In most task-oriented interactions, immersion does not play a major role. It can also be explicitly undesired in some situations, e. g., the interaction with a dialogue-enabled route-planning application in a moving car.

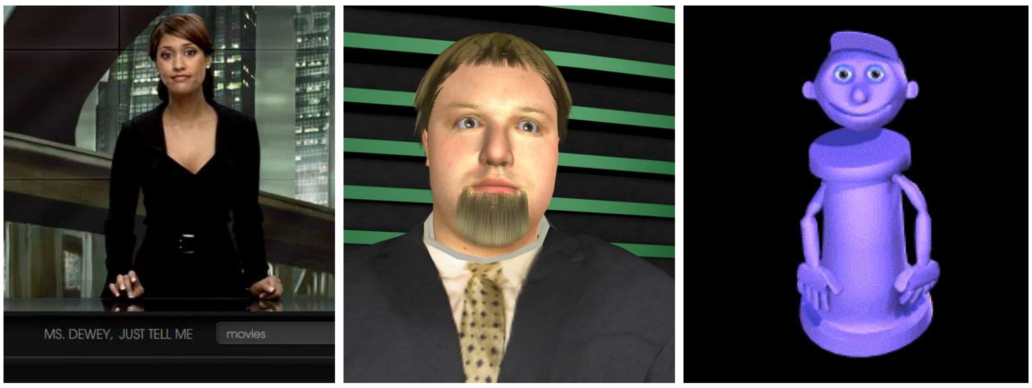


Figure 2.8: Virtual characters in varying degrees of realism: Miss Dewey, Mister Kaiser, and Smartakus

The design of the virtual agents can support or work against the believability of task-oriented and narrative systems. Figure 2.8 shows three types of virtual agents: Miss Dewey, the front-end of Microsoft's prototype search engine *msdewey*[7], Mr Kaiser from *VirtualHuman* (Hülsken et al., 2007), and Smartakus, the system avatar of *SmartKom* (Poller and Tschernomas, 2006). Miss Dewey is animated by concatenating photo-realistic animation snippets performed by a human actor, but her responses are generated using a system similar to the *Eliza* program (Weizenbaum, 1966), i. e., it offers some reactions tailored to specific keywords, and otherwise produces random generic behavior (the search results are displayed in a separate window). Since it is very obvious that the agent does not live up to the expectations raised by her life-like appearance, she is very quickly demoted to pure "eye candy" irrelevant to the search. On the other extreme, confronted with the cute, comic-like appearance of Smartakus, users are positively surprised when they are understood, since their mental model of Smartakus

[7]See *http://www.msdewey.com*

takes into account that it is a "just a computer character". Mr Kaiser is located between Miss Dewey and Smartakus in terms of realism, and also of user expectance of dialogue capability.

2.3.3 Dialogue Designer Perspective

2.3.3.1 Defining the Knowledge Base

The perspective of the designer of the dialogue and the application content is often neglected in dialogue systems research. There are not many frameworks available that are ready to use for authors who are not at home in the fields of linguistics or computer science. This has much to do with the fact that in research, system and content design is frequently done by members of the same team or even by the same person, usually being computer scientists or linguists rather than professional creative writers. Consequently, the available frameworks tend to not be very accessible for non-technical authors, because they require the use of unfamiliar specification methods, like state diagrams or expressions in predicate logic.

In the area of storytelling, systems begin to emerge to address this shortcoming. One example is the *Erasmatron* system (Crawford, 1999) and its successor *Storytron*, designed for writers with non-technical background. The stories created by *Erasmatron* are emergent narratives without goal-based guidance. Also, built-in editors of games like *Quake* or *The Sims* can be employed to create animated stories with game engines (so-called *"Machinima"* movies). The plot of these stories is fixed in advance and allows few (if any) interactions from the user that significantly affect the outcome of the story. Some variation can be added by employing story generators such as the *SceneMaker* tool (Gebhard et al., 2003) that script stories by representing them as scenes connected by conditional transitions in *scene graphs*. This approach was used in, e. g., the *COHIBIT* system. However, these approaches generally do not implement a separation between the overall story structure and the interaction of the story participants.

2.3.3.2 External Control and Narration Engines

One significant difference between pure task-oriented dialogue systems and narrative systems is the need to establish and maintain dramatic tension to keep an audience interested. As said before, the foremost goal for a character in a narrative is not to act *rationally*, but to always remain a *believable* character. The purpose of the narrative is to be entertaining and coherent, but the final resolution of the dramatic tension requires to set narrow boundaries to the freedom of the human interactor as well as of the virtual characters to act. In a typical murder-solving mystery, e. g., it is simply not an option to let the murderer get away in the end; also, in a narrative, the author usually desires a general plot line that must be followed by a successful storytelling system.

For the *VirtualHuman* system, the goal was to keep the story management and the conversation management tasks separate: the conversational engine does not deal with the dramatic constraints of story development. An additional module, the *narration engine*, guides the events by acting in the fashion of a movie director, setting goals for the character agents that are motivated to push the story forward, and to correct intentional and accidental uncooperative behavior from the side of the user. A narration engine uses the available character goals and their parameters as the material from which a narrative can be put together (Göbel et al., 2004). The author must use these as building blocks and select appropriate goals for each situation that can arise during the interaction. With a narration engine setting high-level goals, and the characters having the freedom to determine the exact means for achieving these goals, the system exhibits a *balance of narrative control and autonomy* (Löckelt et al., 2005).

2.3.3.3 Testing and Tuning the System

The number of possible input variations for a typical dialogue system grows exponentially with the length of the interaction, especially if it takes into account the context, i. e., if each conversation turn can in principle be influenced by any previous turn. Generally, it is not possible to test a dialogue manager with all possible conversations, and even limited testing is difficult without full integration of all components of the complete system. This is further complicated, because the surrounding system is comprised of numerous interacting modules; by the nature of its task, the dialogue manager is a very central component in such a system and has to coordinate interactions with many other components. For example, in the *SmartKom* system, the action planner is in direct information exchange with eight other modules (see Figure 2.29 on page 68). Dialogue systems dealing with input that requires recognition and interpretation (i. e., spoken or multimodal input in contrast to typed input) also tend to have a higher response error rate compared to other applications of comparable task complexity because of the significant rate of false recognition results in the more complex modalities such as speech or gestures.

These factors render a complete verification and evaluation of dialogue managers for non-trivial applications very hard or impossible. The quality of a system can be selectively probed with trial runs with groups of users which are rated afterwards using methods for measuring task success as described in Section 2.2.1.3 or more general usability measures as described in, e. g., (Dybkjær and Bernsen, 2001); however, in the general case, such tests are very time-consuming while still only covering only a small part of the possible interaction space. Limited automated analysis of dialogue manager performance as undertaken with, e. g., the *ValDia* system (Alexandersson and Heisterkamp, 2000) or the currently work-in-progress *MeMo* workbench (Jameson et al., 2007) can partly alleviate this effort.

2.4 Dialogue and Conversation Modeling

Besides the overall state of the world the interaction takes place in, the information state must also include the system's model of the conversation. Most research has examined discourses between two participants, which are also called two-party dialogues. Multi-party dialogues (or *conversations*) with more than two participants exhibit additional phenomena and complexities. One example is the issue of deciding which participant has the mandate to speak next after the current speaker releases the initiative (turn taking). However, even in the multi-party case, the interaction proceeds mostly in sequential exchanges involving only two participants at a time, a *speaker* and a (direct) *addressee*. Other participants that may be present—the *overhearers*—can be explicitly or implicitly included. We begin with models concerned with two-party dialogue, and elaborate on additional properties of multi-party dialogue in Section 2.5.2.

Linguistic theory describes several aspects that are essential to capturing a dialogue situation, and must therefore contribute to the information state of the system. One is the context in which the discourse takes place, which includes the mental models of the participants, the social or task situation, and the physical (or virtual) environment. Another is the informational content of the dialogue contributions themselves, and the relations between them. In the general case, a conversation cannot be reduced to a sequence of isolated consecutive statements, e. g., a series of propositional assertions, but it also relies on the relationships between utterances in the discourse context. For example, an answer typically is related to some question occurring earlier in the interaction, and a question in turn imposes an obligation on the addressee to give an answer in the future. Another example is the use of anaphora and ellipses. Besides the linguistic relation between contributions, utterances might also be comprehensible only with respect to the overall context the conversation takes place in.

This means that conversation cannot be comprehensively modeled by looking at the individual dialogue acts alone, but the structure of the whole interaction needs to be taken into account. A collection of unrelated utterances does not constitute a proper conversation, it has to be *coherent* to be meaningful. The coherence relations of a conversation are however derived from the basic dialogue acts and their communicative functions.

2.4.1 Speech and Dialogue Acts

(Austin, 1962) introduced the notion of spoken utterances being a kind of action, calling them "dialogue acts". Searle later elaborated on this theory more formally in (Searle, 1969) and together with Vanderveken in (Searle and Vanderveken, 1985). The correspondence of a dialogue act to its effect on the world can be immediate and direct, as in performative utterances such as christening a ship or declaring war, or less direct, for example as in the sentence "It's cold in here", whose intended effect could be solely to confer a belief or opinion about the state of the world to a listener; it might however—within certain contexts—pragmatically cause somebody to turn on a radiator or close a window, and be intended to do so. Bunt gives the following definition of a dialogue act:

> *"A dialogue act is a unit in the semantic description of communicative behavior produced by a sender and directed at an addressee, specifying how the behaviour is intended to influence the context through understanding of the behaviour."* (Bunt, 2005, p. 2)

The notion of *intended context change* directed at an addressee is central. It stresses that not only the form of the utterance itself is important, but also its (expected) interpretation by the addressee. To be able to know, or at least presume, what the sender will effect by a producing a dialogue contribution, it is necessary to have some understanding of the way the addressee is going to analyze it. The addressee-dependent context that is referred to in Bunt's definition is an informational account of the conversation as well as the general situation in which the discourse takes place. The conversation participants generally cannot be expected to have identical context representations. They can differ because the participants have different beliefs and attitudes about the world.

Bunt's definition implies that, to understand how utterances in a conversation are produced and understood, the following questions must be addressed:

- what kinds of context changes can be effected by dialogue acts, and
- in what way can a conversation participant anticipate how an utterance will change the context for an addressee?

In the context of dialogue systems, different types of context change will result in corresponding updates of the information states of the interlocutors, as well as possibly the representation of the environment.

2.4.1.1 Types of Context Changes

The mapping of linguistic signs to their intended context change, and with it their meaning, are not inherent, but determined by pre-agreed *conventions* between the interlocutors. Also, the "protocol" of conversation is governed by rules. A participant in a conversation needs to know and adhere to these conventions and rules to be able to successfully understand utterances of others, and make his own contributions to a conversation (Wittgenstein, 1953).

Frequently, the type of intended context change of a contribution is included in the form of the utterance itself. For example, in the case of questions, in addition to marking them by question cue words ("When *will John come?"*), questions can be distinguished from statements by way of syntax (*"John is coming tomorrow"* vs. *"is John coming tomorrow?"*). A verbatim statement can also be transformed to a question *"John is coming tomorrow?"* without syntactical change by cues in other modalities, such as prosody or facial expression. The context can also determine the type of speech act, e. g., the utterance "yes" can, e. g., be a statement about some fact, or a commitment to do something, depending on previous utterances.

Dialogue acts can have effects on several levels. Austin distinguishes between three facets of action in an utterance:

- the *locutionary act*, which is the physical action of producing an utterance,
- the *illocutionary act*, which is the speaker's intended meaning of the utterance, which could be a statement or a request. According to Searle, The illocutionary act has the further aspects of illocutionary context (the environment in which the utterance is to be understood), its propositional content, and its illocutionary force.
- the *perlocutionary act*, which is the action that results from the utterance, e. g., getting the addressee to answer a question, or to incorporate the content of a statement into her set of beliefs.

(Searle, 1975) elaborates on Austin and gives five major classes for speech acts:

- *assertives* – statements of facts, expressing beliefs of the speaker
- *directives* – commands, requests, questions, etc.
- *commissives* – promises, offers, etc.
- *expressives* – expressions of feelings and attitudes, e. g., apologizing or greeting

- *declaratives* – acts that themselves perform an action, e. g., naming something

(Pedersen, 2002) points out that these classes can be further subdivided to yield more fine-grained speech act types, as in (Bach and Harnish, 1962). He gives rich examples for verb groups that each capture different shades of meaning for the types. His examples for English verbs describing, e. g., assertives include: *affirm*, *assert*, *claim*, *declare*, *say*, *state*, *submit*, *forecast*, *predict*, *ascribe*, *attribute*, *categorize*, *describe*, *identify*, *judge*, *testify*, *conclude*, *agree*, *guess*, and many more. Also, (Pedersen, 2002) marks as important aspects of the meaning of dialogue acts the notions of belief, want, desire, intention and obligation. Each of these notions can apply to the speaker as well as to the listener in that, e. g., the utterance of an act may impose obligations on either of them, or on both.

A further complication is that a speech act may be intended to convey pragmatic content that is very different from its *surface* (or "literal") meaning, depending on the context and inferences that are implicitly assumed by the speaker. Consider the following utterances:

(1) *Can you open the door for me?*

(2) *You are Peter Miller, I assume.*

(3) *You are* so *right.*

(4) *That was very wrong of me.*

Utterance (1) usually does not constitute a request for information, but is a polite way of instructing somebody to open the door. (2) is a question in the surface form of a statement. It is possible (but not necessarily clear out of context) that (3) is a sarcastic statement, as hinted at by the emphasized keyword *so*, and means just the opposite to what is actually said. Finally, (4) is an utterance that can simultaneously be interpreted as a statement and an apology. The last example also illustrates the important point that there not necessarily has to be a clear one-to-one mapping from utterances to speech act types. There are many possibilities to formalize the semantics of speech acts. Most use some sort of logic, either first-order logic or modal logics that additionally model concepts of, e. g., temporal constraints or necessity (cf., e. g., (Gamut, 1991)).[8]

2.4.1.2 Dialogue Acts

In the context of dialogue systems, the wider concept of *dialogue acts* (also called *conversational moves*) enriches speech acts with additional functions in the context of conversations. Some examples are (Traum and Hinkelman, 1992), (Bunt, 1994), the dialogue coding scheme *DAMSL* (Core and Allen, 1997; Carletta et al., 1997) which has later been expanded for multimodal acts (Pineda et al., 2000), and the analysis of dialogue acts in the *Verbmobil* project (Alexandersson et al., 1995).

The exemplary taxonomy shown in figure 2.9 builds on the work in the *TRAINS* project and *DAMSL*. It specifies a set of task-independent dialogue acts derived in the Discourse Resource

[8]see (Pedersen, 2002) for a more extensive summarization of the development of speech act theory.

- Core Speech Acts
  - Forward looking acts

    ```
    Statement
      Assert, Reassert, Other-Statement
    Influencing-addressee-future-action
      Open-option
      Directive
        Action-directive, Info-Request
    Commiting-speaker-future-action
      Offer, Commit
    Conventional
      Opening, Closing
    Explicit-performative
    Exclamation
    ```

  - Backward looking acts

    ```
    Agreement
      Accept, Accept-part, Maybe, Reject, Reject-part, Hold
    Answer
    ```

- Grounding Acts

  ```
  Understanding-act
    Signal-non-understanding
    Signal-understanding
      Acknowledge, Repeat-rephrase, Completion
    Correct-misspeaking
  ```

- Turn-taking acts

  ```
  Take-turn
  Keep-turn
  Release-turn
    Assign-turn
  ```

Figure 2.9: A taxonomy of dialogue acts (Poesio and Traum, 1998)

Initiative, with special attention to grounding and turn-taking acts. The set is hierarchically structured in terms of classes and subclasses of actions, where subclasses inherit the properties of a superclass. The class of *core speech acts* that are used to "manage the topic of the conversation in a general sense" (Poesio and Traum, 1998) is further subdivided into acts with a *forward looking* and *backward looking* function, i. e., they relate to other acts in the future or the past conversation, respectively. Further classes are acts used for *grounding* purposes, and *turn-taking*. Again, a single locution may constitute an instantiation of more than one dialogue act type.

There is no proven set of dialogue acts, or one that is de facto generally agreed-upon in the research community. Neither is there a canonical taxonomy to define the relations between acts. Some important open issues in this field are (Traum, 2000):

- what distinguishes genuine dialogue acts and other (e. g., physical) acts, communicative or otherwise, and how can different kinds of acts be related if they occur together in a conversational situation?

- how are dialogue acts related to the dialogue structure beyond the scope of individual utterances, and how do they extend to multi-party dialogue (e. g., are there multi-agent dialogue acts)?
- which classes of dialogue acts should be distinguished, and what is their precise semantics and role in the dialogue?
- what formalism should be used to represent dialogue acts, and how can one avoid sacrificing common-sense intuitions or desirable formal properties?
- can the same taxonomy be used for different kinds of dialogue? How detailed should a dialogue act taxonomy be?

It is therefore still an unsolved problem, and may well also be infeasible altogether, to determine a general set of dialogue acts suited to every kind of human communication, or dialogue, or a single canonical taxonomy or representation for them. Fortunately, the fact that these linguistic problems are not yet resolved does not prevent us from creating our framework for dialogue management. The approach we describe in Chapter 5 is not dependent on such an agreed-upon set of dialogue acts. For our framework, it is not required to commit to a specific dialogue act categorization. We will instead use a basic dialogue act taxonomy similar to the ones of (Poesio and Traum, 1998) and (Alexandersson et al., 1995) that is meant to be extended by introducing additional subtypes if and when it is convenient or necessary for a specific application (see section 5.6.1).

2.4.2 Discourse and Dialogue Structure

2.4.2.1 The Hierarchical Structure of Discourse Segments

Above the level of dialogue acts, a coherent conversation is characterized by higher levels of structure. This includes the relations between the utterances as well as the mental models that are constructed by the participants as the conversation progresses. Both contribute to the meaning and understandability. (Grosz and Sidner, 1986) distinguish three main components in discourse structure: The *linguistic structure* of the sequence of utterances, the *intentional structure*, and the *attentional state* of the participants.

The segments of a conversation comprising units of linguistic structure, usually one or more utterances per *turn* where a participant makes a contribution, can be grouped in *discourse segments*. These segments can be hierarchically ordered according to their role or *discourse purpose* in the conversation, as in, e. g., a train ticket sale interaction represented in figure 2.10. The intentional structure accounts for the discourse purpose of each segment by assigning corresponding *discourse segment purposes* (DSPs). A DSP might be, e. g., to make some fact known to a dialogue partner, or to persuade a dialogue partner to do something. The figure does not make explicit whether each segment consists of just one, or many utterances. A DSP structure could also be seen as a *recipe* for the structure of such a corresponding dialogue. Each segment can also possibly be composite, and recursively decompose into additional subsegments to include, e. g., clarification exchanges when necessary.

Grosz and Sidner propose the coherence relations of *dominance* and *satisfaction-precedence* between DSPs. A DSP A dominates another DSP B if B provides a part of what is necessary

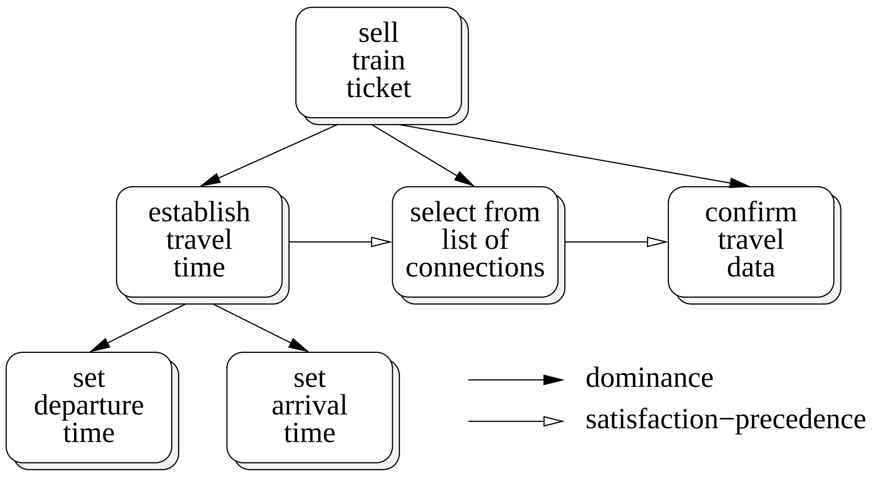


Figure 2.10: DSP relations in a fragment of a train ticket sale dialogue

to establish A's purpose, and A has satisfaction-precedence over B if A must be satisfied before B can be satisfied. The train ticket sale example in figure 2.10 illustrates this: the DSP *sell train ticket* dominates the DSPs that provide travel times, a connection selection, and a confirmation from the user, since a train ticket cannot be sold without establishing this information and getting the user to agree to it. A suitable list of connections can only be presented after the travel times have been set, therefore *establish travel time* satisfaction-precedes *select from list of connections*. Since departure and arrival time can be set in any order (or possibly just one of both might be necessary), there is no relation of dominance or satisfaction-precedence between the corresponding DSPs. Both coherence relations are transitive and irreflexive (the transitive closure for the relations is not shown in the figure).

(Allwood, 2000) holds that the use of linguistic expressions is defined by grammatical structure, communicative function and their occurrence in a *social activity*, which is characterized by the parameters

1. type and and function (*purpose*) of the activity,
2. *roles* that define the competence, obligations, and/or the rights of the participants in the activity,
3. *instruments* used in the activity, and
4. other physical environment

If discourse segments are interpreted as sub-activities in a larger social activity governing the overall discourse, and DSPs are seen as the purposes of sub-activities, the DSP relations can already give some idea of how an agent could go about planning the sequence of communicative actions necessary to achieve a task.

2.4.2.2 Discourse History and Context

To understand an ongoing discourse, the history of past utterances and other context, linguistic and otherwise, has to be considered.

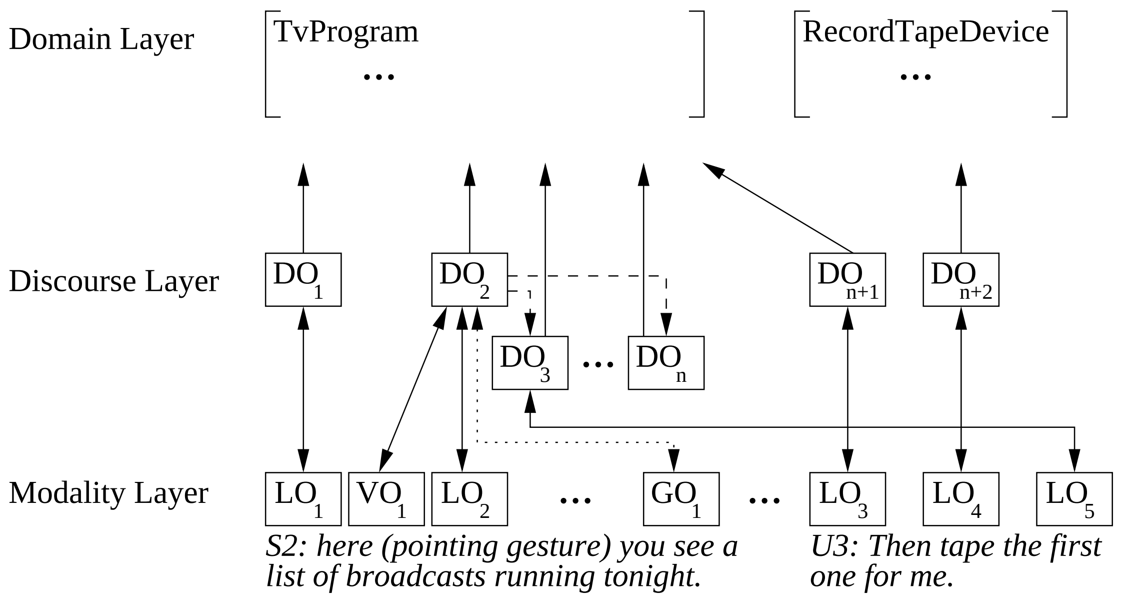


Figure 2.11: Example multimodal context representation (simplified, adapted from (Pfleger, 2007))

The recording of the course of the conversation is called the *discourse history*. In all but very simple systems, this information does not only comprise an unordered collection of facts about the interaction, but is also organized hierarchically according to the discourse and/or task structure.

Knowledge of previous contributions and the context in which they occurred is necessary to resolve elliptical utterances. It can also provide cues for the correct meaning of ambiguous contributions and enable other instances of context dependent interpretation. An example is the fusion of multimodal contributions, which can benefit enormously from close integration with the discourse history. Figure 2.11 shows a simplified example of the dialogue history representation used in the *SmartKom* system that also accounts for multimodal contributions. It comprises three layers: the modality layer, the discourse layer, and the domain layer. For dialogue systems, it is a design decision whether the discourse history should be stored and processed separately from the task state (Allen et al., 2001b).

2.4.3 Information State

To engage in a dialogue in a meaningful way, the participants need to have some understanding of what is going on in the utterances, as well as how they refer to the context in which the conversation takes place. That means that the dialogue management components have to maintain a knowledge base that is commonly known as the *information state* of the system.

The necessary complexity of the information state for different dialogue systems spans a wide range, depending on the complexity of the interaction as well as the underlying application.

On the side of simplicity, there are systems that are essentially finite state automata. The majority of current automated phone-based services are examples of this. Using cue word recognition, they frequently present the user with a voice-navigable menu system that is able to answer common requests and tasks, or narrow down the topic area and route the caller to a specialized human operator. The finite state approach is also suitable for other tasks that can be modeled by a limited number of contextual "frames" or "forms" to be filled in, corresponding to application stages or states, in a fixed order.[9] In this case, the state of the dialogue manager essentially coincides with the current state the automaton is in, possibly extended by some account of the interaction history in terms of filler values.

On the other hand, more elaborate interaction paradigms require additional information to model the task state, the state of the interaction, and the states of the interactors in user and character models that account for the beliefs, motivations and plans of the interactors. Such systems make use of a more powerful and comprehensive representation of the state of the world and the actions the system is capable of, e. g., in terms of logical descriptions. Such a representation allows more powerful and flexible reasoning using, e. g., theorem provers and planning mechanisms. It imposes however, as a trade-off, increased demands with respect to computational complexity and storage requirements; e. g., inferences using fully-fledged first order logic theorem proving is semi-decidable and thus cannot guarantee reasonable time bounds for system responses in the general case. To avoid this problem, dialogue systems often resort to knowledge representation languages using *Description Logics* that offer decidability in exchange for different subset restrictions of first-order logic with varying degrees of expressivity that disallow, e. g., negation or existential quantification. The first such language was *KL-ONE* (Brachman and Schmolze, 1985). Modern ontology languages for applications in the semantic web, e. g., *OWL-DL*, also are based on description logics.

Between plain state-based and fully-fledged logical representation there is a wide variation of paradigms. The following sections describe the approaches the Belief-Desire-Intention model, the *SharedPlans* model for collaborative action, and the information states of the *TRINDI* and *SmartKom* systems.

2.4.3.1 The Belief-Desire-Intention Model

The *Belief-Desire-Intention (BDI)* approach was introduced by Rao and Georgeff (Rao and Georgeff, 1991), which expanded on earlier work by Cohen and Levesque (Cohen and Levesque, 1990) and Bratman's theory of human practical reasoning (Bratman, 1987). It describes the motivations for action in a rational agent by interpreting the folk psychology notions of beliefs, desires, and intentions in terms of computational agents.

The *beliefs* of a BDI agent are the set of logical assertions in that particular agent's knowledge base that the agent assumes to be true. This does not actually have to objectively be the case. The belief set can also include inference rules that allow one to generate new beliefs. For a rational agent, the set of beliefs should be logically consistent.

[9]Some flexibility is added if the states can be ordered dynamically, like in e. g., *VoiceXML* systems.

The *desires* of an agent are motivations towards goals that the agent wants to achieve without having decided to actively take action to pursue them. They can be abstract in the sense of establishing conditions to "be happy", or rather concrete, like achieving some goal. An agent having particular desires does not necessarily mean that actions to satisfy them have to be "on its agenda" at all times; they can be suspended if others take precedence.

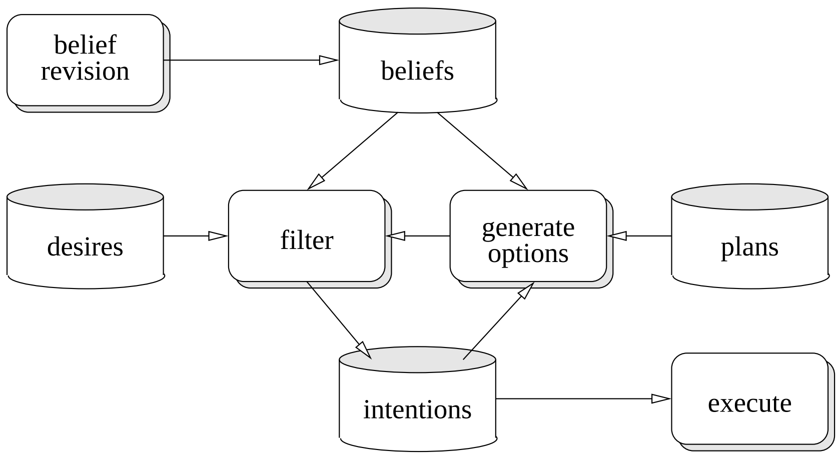


Figure 2.12: Practical reasoning in a BDI agent (adapted from (Weiss, 1999))

An agent's *intentions* represent what it has decided to do to reach the goal to satisfy one or more desires that are attainable in a given situation, e. g., in terms of instantiated plans or plan recipes. There may be more than one possible way to reach such a goal, in which case the agent has to make a selection between them. Once an agent has decided to adopt a certain plan, its execution becomes an *intention* of the agent; it can be said that intentions therefore represent "choice with commitment" (Cohen and Levesque, 1990). The choice can be guided by preferences of the agent, e. g., assumed *utility* values for the different alternatives. While different desires of an agent can conceivably be and remain in conflict with each other (e. g., it can be rational to simultaneously *desire* to go to work and stay in bed), however, the set of simultaneously adopted intentions should be consistent for a rational agent (i. e., one cannot rationally *intend* to go to work and stay in bed at the same time). (Georgeff et al., 1999) point out that to have intentions that are derived from desires gives a reason, or explanation, for why those actions are done by a computational agent, and this constitutes a difference to "conventional" program execution:

> *"... a goal represents some desired end state. Conventional computer software is "task oriented" rather than "goal oriented", that is, each task (or subroutine) is executed without any memory of why it is being executed. This means that the system cannot automatically recover from failures (...) and cannot make use of opportunities as they unexpectedly present themselves."* (Georgeff et al., 1999, p.4)

As an extension of BDI, it has been proposed to also include the *obligations* of an agent, resulting in a "BOID" model. Obligations are externally imposed constraints or requirements on actions that are imposed on an agent by way of social conventions, norms, or commitments. (Broersen et al., 2001) examine the conflicts that arise from the combination of the four components of BOID as well as possibilities to resolve them. Obligations can be violated, e. g., if the desire to do something is stronger than the obligation to refrain from doing it. The priority balance between external obligations and internal desires of the agent can also be seen as a measure of how "socially behaved" the agent is. This allows the construction of different personality types of agents that exhibit different preferences with regard to the resolution of conflicts between beliefs, desires, intentions, and obligations; e. g., an agent is called "selfish" if it tends to let desires override obligations, or "social" in the opposite case (Broersen et al., 2001).

Implementing computational BDI agents for complex domains has turned out to be difficult. One problem is that, e. g., fully rational action based on a BDI model would require oversight of all consequences of one's beliefs and actions, which is generally not practical for nontrivial settings (problem of logical omniscience).

2.4.3.2 *SharedPlans*

SharedPlans theory (Grosz and Sidner, 1990) formalizes the conditions under which a group of agents can be said to have a shared plan to collaboratively achieve some objective. The theory was implemented in, e. g., the *COLLAGEN* system (see Section 3.2.2), but continues to be developed.

The theory states that agents have (shared) plans when they jointly hold a certain set of intentions and beliefs, i. e., it assumes a mental-state view of plans. There is a distinction between having *recipes* and having *plans*. A recipe for an action α is a description of how to do α, while a plan is an instantiated recipe that represents the intention to do α in a certain way. Agents in collaboration share mutual beliefs and plans. Mutual belief in a statement σ means that, in addition to each agent believing σ, each agent believes that all other agents believe σ, and so on. A summary description of the conditions for a group G to hold a shared plan for action α using a recipe R_α is (Grosz and Kraus, 1996):

(1) All members of G have mutual beliefs in a recipe R_α leading to the group success of doing α.

(2a) Each member of G intends that R_α is done,

(2b) Each member of G intends that the collaborators succeed in doing the constituent sub-actions,

(3) For each sub-action β in R_α, there is an agent or sub-group G' which has an individual or shared plan to do β, and everyone else in G must believe that G' can do the sub-actions using an appropriate recipe (it is not required that other agents know the recipe).

Note the distinction between *intend-to*, i. e., the intention to (personally) do an action and *intend-that*, the intention to (possibly jointly) bring about some state of the world. The conditions can be elaborated for, e. g., *partial* shared plans in which not all steps are (yet) fully specified. They include additional items in (1) concerning the mutual belief of the members of G that all participants are committed to identify the parameters for R_α and satisfy its constraints, and (3) can be extended to cover different cases of deliberation and conflict resolution (see, e. g., (Grosz and Kraus, 1999)). Agents collaborating on some goal can use operators to build shared plans (Grosz and Kraus, 1996):

- *selecting:* from an available set of recipes to accomplish a subgoal, an agent or a group of agents select a recipe using the operators *Select_Rec* or *Select_Rec_GR*
- *elaborating:* an agent or a group of agent decompose a selected recipe into subactions using the operators *Elaborate_Individual* or *Elaborate_Group*.

Using these operators, goals are decomposed until the level of fully specified atomic actions is reached. An algorithm to build *SharedPlans* has been given in (Lochbaum, 1998).

The *SharedPlans* formalism focuses on the process of building a shared plan between agents and does not account for the actual process of execution (Blaylock, 2005); some issues concerning the underlying reasoning and, e. g., commitment rules also are not fully specified (Grosz and Kraus, 1996). The formalism also does not integrate plan-building with other communicative actions the agents perform. (Nguyen and Wobcke, 2005) points out that the available implementations are hard-coded for a specific application, and a full framework implementation of the theory for more general domains has not yet been constructed.

2.4.3.3 *TRINDI* Information State

TRINDI-based systems use an information state that is based on Ginzburg's notion of a *dialogue game board* (DGB) that represents the dialogue information publicly shared between participants (Ginzburg, 1996). It is combined with a private information state for each participant (representing Ginzburg's unpublicised mental state). Both together form the structure called the "total information state". The total information state keeps track of a dialogue by successive updates triggered by the moves of the participants. It can then be used to motivate and trigger future action.

$$\left[\begin{array}{ll} \textsc{Private} & \left[\begin{array}{ll} \textsc{Agenda} & \mathit{Stack}(\textsc{Action}) \\ \textsc{Plan} & \mathit{Stack}(\textsc{Action}) \\ \textsc{Bel} & \mathit{Set}(\textsc{Prop}) \end{array}\right] \\ \textsc{Shared} & \left[\begin{array}{ll} \textsc{Com} & \mathit{Set}(\textsc{Prop}) \\ \text{QUD} & \mathit{Stack}(\textsc{Question}) \\ \text{LU} & \left[\begin{array}{ll} \textsc{Speaker} & \mathit{Participant} \\ \textsc{Move} & \mathit{Move} \end{array}\right] \end{array}\right] \end{array}\right]$$

Figure 2.13: A *TRINDI* information state

Figure 2.13 shows an example from the *IBiS1* system (Larsson, 2002). The public part (SHARED) stores information about the current question under discussion (QUD) in the dialogue, propositions the speaker and the system have mutually agreed to (COM), and information about the latest utterance (LU). The PRIVATE part contains the agenda of the system agent (AGENDA) as a stack of actions, a set of plan constructs (PLAN), and a set of beliefs held by the agent (BEL).

An information state theory consists of (Larsson and Traum, 2000):

(1) a description of the informational components of the dialogue model, including context and motivating factors such as participants, user models, obligations, beliefs, etc.,

(2) formal representations of the informational components in terms of, e. g., lists, discourse representation structures, or logic expressions,

(3) a set of *dialogue moves* that realize natural language utterances and trigger information state update,

(4) a set of *update rules* that are applicable depending on the current information state and performed dialogue moves,

(5) an *update strategy* for deciding which rule to select from a set of applicable ones, which can be a simple "pick the first one" strategy, or use more sophisticated arbitration mechanisms, such as game theory, utility theory, or statistical methods.

A *TRINDI* dialogue manager executes an update loop that drives the dialogue by selecting and applying appropriate update rules and executing associated dialogue moves.

2.4.3.4 *SmartKom*'s Information State

In the *SmartKom* system, the main information state is split between two modules, the *discourse modeller* (DiM) and the *action planner* (AP).[10] The discourse history module models the structure of the interaction with the user in terms of dialogue segments in a three-tiered model. DiM's information roughly corresponds to the SHARED part of the information state in *TrindiKit*-based systems (the progression of the interactions between the action planner and the application modules is not stored in the dialogue history). Not only spoken contributions, but all multimodal interactions are stored, since the user can also refer to past non-spoken contributions in, e. g., anaphora (Pfleger et al., 2003).

The action planner holds the structures that contain the task-related domain knowledge, intentions and discourse-independent beliefs of the system. This in turn corresponds to the PRIVATE part of the information state. For each active application, a *discourse object* is maintained that contains the data relevant to the application (Figure 2.14 shows a discourse object representing an information search action in a route-finding application). Underspecified discourse objects are also used to represent the content of dialogue contributions related to the application. Discourse objects can be passed between applications that work together; in some cases, they can also be converted to other data types, e. g., when the dialogue is about scheduled cinema performances and switches to the TV program, parameters such as time restrictions and genre preferences are adapted and retained in the discourse object for the new application (Porzel et al., 2003; Alexandersson et al., 2004b).

Alternative hypotheses for the meaning of incoming user intentions are annotated with scores at the different processing stages of speech recognition, speech interpretation and media

[10]The run-time data some other modules maintain, e. g., the dynamic lexicon and the help module, also affect the interaction and could be seen as part of the overall information state of the system; however, the flow of the main interaction is determined by the states of AP and DiM.

$$\begin{bmatrix} \textit{InformationSearch} & \\ \textsc{pieceOfInformation} & \begin{bmatrix} \textit{Sight} & \\ \textsc{location} & \begin{bmatrix} \textit{LocationType} & \\ \textsc{objectKey} & \begin{bmatrix} \textit{ObjectKeyType} & \\ \textsc{key} & \textit{147} \\ \textsc{type} & \textit{“spatialObject”} \end{bmatrix} \\ \textsc{locationName} & \textit{“Schloss”} \\ \textsc{objectTypes} & \langle \textit{Castle} \rangle \\ \textsc{geometries} & \left\langle \begin{bmatrix} \textit{Point} & \\ \textsc{x} & \textit{3479352.0} \\ \textsc{y} & \textit{5474910.0} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 2.14: An *InformationSearch* discourse object from a *SmartKom* information state

fusion. The discourse modeller performs context-dependent semantic enrichment on these hypotheses, e. g., reference resolution, which also adds a score to each. After that enrichment, an *intention recognizer* module selects one hypothesis according to a weighting of the different scores, and passes it on to the action planner.

The action planner manages a stack of discourse objects that belong to the active applications and are dynamically updated during the interaction. When an active application is completed, it is either possible to either continue a previous application that was incompletely executed before the current one (whose discourse object is located below in the stack), or to start a new unrelated application (cleaning the stack). The first option allows applications to execute functionalities of other applications as sub-procedures. However, *SmartKom* does not incorporate more sophisticated switching between tasks, and especially does not allow to execute more than one task simultaneously.

Contrary to *TRINDI*'s information state, the information state in *SmartKom* does not include rules for updates of discourse objects or the selection of actions (items (4) and (5) in the list in the previous section). This information is represented separately in a set of *action plans* that combine declarative and procedural elements to define a set of possible dialogue games for each application in *SmartKom*. A custom planner combines the information in the following way: it uses the action plans as operator specifications, and the information state as a description of a task state for a planning problem. The operation of *SmartKom*'s action planner is described comprehensively in (Löckelt, 2006).

2.4.4 The Impact of Multimodality

2.4.4.1 Rationale

In everyday communication, humans generally uses multiple modalities. On first glance, a multimodal dialogue system therefore promises to be more intuitive to use. There are however additional benefits. The potential as well as possible misconceptions regarding the use of multiple modalities were analyzed, e. g., using the example of the *QuickSet* system. Aside from more natural interaction, they provide a greater robustness of recognition, and superior error avoidance and graceful recovery from errors (Oviatt, 2002), greater expressive power and efficiency, and better accessibility for users with different skill levels, native languages, cognitive styles, and sensory or motor impairments (Oviatt and Cohen, 2000). Oviatt rejects the assumption that a combination of different error-prone recognition technologies might result in greater unreliability, arguing that instead the input can be combined to remedy errors and ambivalent input through mutual disambiguation offered by redundant or complementary data (Oviatt, 1999).

Key issues for multimodal interaction are systems with multiple (not just two) modalities, symmetric multimodality (i. e., balanced multimodal input and output) and the integration of multimodality with virtual environments and synthetic characters (Wahlster, 2003b; Oviatt, 2002). For systems featuring non-trivial interactive communication with virtual characters, called “embodied interfaces” by Cassell, the ability to flexibly produce (and recognize) contributions in different modalities becomes a necessity (Cassell et al., 1999; Cassell, 2001).

Common input modalities for interaction with dialogue systems include speech, gestures, facial expression, gaze, as well as “conventional” mouse interactions with GUI components, and of course typed input. There are also systems where the user can interact via manipulations of the physical environment, such as moving objects (e. g., the *COHIBIT* installation (Ndiaye et al., 2005) lets the user assemble a car prototype from different car parts, and reacts to positioning the parts in different physical places), or handling physical avatars (e. g., the “emotion transmitting doll” mentioned in Section 2.3.1). Many types of multimodal output require that the system features some kind of “embodiment” to carry the message, usually in form of an (abstracted or realistic) avatar of the system. In some cases, modalities cannot be straightforwardly symmetrical (e. g., graphic display of data by the system).

2.4.4.2 Processing

Although we are concerned with multimodal systems, we assume that the integration and disambiguation of multimodal input, as well as the distribution of output into suitable modality combinations are the tasks of two components separate from the dialogue manager: A *modality fusion* component constructs a unified input representation from the output of the recognizers for the different input modalities. Conversely, a *modality fission* component decomposes output in this combined representation and yields component acts that can be distributed to different generation components.

This means that the input and output of the dialogue manager can be in terms of *multimodal communication acts* that combine the content of all modalities in a unified semantic representation. With dedicated fusion and fission components present, the dialogue manager can remain largely “modality-agnostic” with respect to its input and output, as the units of communication are multimodal “discourse objects” (see Figure 2.11). However, some issues related to multimodality remain that must be considered, for example

- **Input and Output Synchronization**

  Frequently, multimodal input occurs in a time-overlapping manner, or with delays. An example are utterances accompanied by pointing gestures, as in the following:

  > USER: *Select this* [↗ (player$_4$)] *football player.*[11]

[11]We include pointing gestures by a notation of “[↗ (referenced object)]” in transcriptions.

USER: *Put Ballack there* ... [hesitates] [↗ (position$_7$)]

Combined with the fact that recognizers for different modalities may take different processing times, it follows that it is not always possible to pass on an utterance immediately when it has been recognized. Instead, multimodal fusion components often set some time frame to wait before proceeding in case the contribution is not (yet) complete. This however can impact the responsiveness of the system negatively. The generation of multimodal output also requires synchronization. A constraint-solving component can be employed to generate a schedule for the production of a multimodal act (e. g., the *ActionEncoder* described in (Klesen and Gebhard, 2007)). There are also some questions with regard to when a contribution is actually considered to have begun or finished from the point of view of the participants; this issue will be discussed in more detail in Section 7.2.4 on realization scheduling.

- **Presentation Planning**

  In some cases, the timing of the system contributions and their distribution among different modalities is partly or completely done by the dialogue manager. This is sensible in cases where there is no dedicated presentation planning component, or where the presentation component is a stateless module altogether.[12] In this case, a unified representation allows for selection of different modality combinations for the same multimodal presentation depending on context. For example, if the system is used in a mobile environment such as a moving car by the driver, in some situations it can be inferred that it is not appropriate to use a graphical display since the user must not divert her attention from the street while driving. In this case, it is better to choose a purely speech-based rendering of the output. Another example is presentation planning to manage limited screen estate on smaller devices. If the system is reacting to a multimodal user utterance, information about the source modality, if retained in the unified input representation, can also be exploited to aid in the selection of appropriate output modalities. It can also be beneficial if the dialogue manager can determine or at least influence the selection of output modalities, e. g., to accommodate the preferences of the user, or environment-related constraints (e. g., using non-speech modalities in a noisy situation). Along these lines, to enable the dialogue manager to be responsive to user modality choices, the multimodal communication acts should retain information about the original modalities the different parts of the input were recognized in.

The systems that have been realized with the framework from this thesis, described in Chapter 7, all feature multimodal input and output. In all cases, a designated module (*FADE*) is assigned to do modality fusion, but the task of modality fission is taken over by the dialogue manager.

2.5 Dialogue Management

The last sections concentrated on descriptions of the elements and structure of dialogues, as well as the mental states of the participants. This section now looks at dialogue from

[12] This is the case for all use case examples described in this thesis.

the perspective of how the participants interact in it as a *joint activity*, and how a dialogue manager may go about to partake such an activity in controlling one or more participants. We also examine several relevant phenomena that arise during interactions with dialogue systems, and what is usually done about them. We also look at the additional qualities and requirements introduced in multi-party dialogues, i. e., dialogues involving more than two participants.

All parties involved in a dialogue must satisfy essential constraints to "play by the rules". First, even when ignoring task level cooperativity, for communication to have any chance to be successful at all, there has to be at least a basic level of interaction cooperativity. If participants do not coordinate their usage of the available channels used to exchange communication signals, e. g., by means to avoid simultaneous speech, understanding will be difficult or impossible for everyone. Also, in a dialogue, the participants can generally be said to have individual, joint, or possibly conflicting *goals* they pursue. In a task-oriented interaction, it is usually in everyone's interest to act cooperatively and solve the task together. In a typical interactive narrative, every character has personal goals, and those of the antagonist of the player will—*per definitionem*—include to work against the player. However, cooperativity on the dialogue level is more fundamental. Unless they want to cause utter confusion, even openly hostile dialogue partners are bound by the basic rules of dialogue; otherwise, it might be just as useful to have no conversation at all.

In the next sections, we examine some aspects of communication as joint action. First, look at patterns underlying communicative exchange. Second are issues of negotiating the right to speak at a given time. Afterwards, we treat how the respective information states of the dialogue participants are accommodated to ensure mutual understanding and agreement of what the dialogue is about.

2.5.1 Patterns for Communicative Exchange

As stated in section 2.4.2, related utterances group into discourse segments with certain purposes. We now look at *how* this grouping can be modeled, where, like with syntactic phrases and their constituents, *"the meaning of a segment [is] encompassing more than the meaning of the individual parts"* (Grosz, 1997). The interactive patterns are important for dialogue management from different viewpoint angles.

- *Descriptive:* In what way can rules or patterns be specified that set apart sequences of utterances that are meaningful / rational / cooperative from ones that are not?
- *Interpretative:* Given the meaning of the constituents and the context, what is the (pragmatic) meaning of each utterance?
- *Generative:* How can a dialogue manager generate appropriate utterances?

Prominent approaches in this respect are the *dialogue grammar*, *dialogue games* and *plan-based* approaches, which we will concentrate on here (in Chapter 3 we also give some examples of differing approaches used in deployed dialogue systems). To summarize a comparison made in (Cohen, 1997), the dialogue grammar approach is based on the observation that there are *adjacency pairs*, like “question-answer” or “proposal-acceptance”, where the first element of a pair generates an *expectation* for the second, and violations of such expectations are apt to disrupt the dialogue. Dialogue grammars use rules similar to syntactic phrase-structure grammar rules, where terminal rules correspond to dialogue act representations, and can be used to predict possible next acts given a prior sequence. The related dialogue games approach, which will be further explained in the following section, also groups utterance elements together according to a set of rules to generate predictable sequences, but sees them as *moves* in a game between the participants. This offers the possibility to identify expected next moves, like in other games. The plan-based approach starts from the underlying task structure and sees the utterances of the participants as means to convey their intentions in addressing a (joint) task. Planning and plan recognition algorithms can then be employed to infer appropriate subsequent utterances.

In our framework, we follow the proposal made by (Hulstijn, 2000a) and use the dialogue game and plan-based approaches in a complementary fashion to model dialogue as a process of joint action using shared schematic (sub)plans—or, “recipes”—derived from the patterns provided by the rules of the former. As Hulstijn put it in the title of his paper, *“Dialogue Games are recipes for joint action.”*

2.5.1.1 Dialogue Games

The notion of seeing the interaction in a dialogue as a sort of game has a long history, reaching back as far as Aristotle, who investigated rules for argumentation in his work *Topics* (Aristotle, 1928). In the 20th century, linguistic research took up the notion again. Wittgenstein viewed the meaning of language to be determined by way of *how it is used* in an exchange between humans, and introduced the term “language game” in a series of thought experiments about the nature of human communication (Wittgenstein, 1953). In such games, the use of non-linguistic devices, such as pointing, was also included to convey meaning.

Later research, especially in the field of computational linguistics, found the game metaphor useful, not least because the availability of explicit rules for dialogical interactions facilitates an algorithmic approach. Early work in this area is (Schegloff and Sacks, 1973) examining conventions that exist between human dialogue partners regarding the termination of conversation. Such conventions need not involve making actual utterances, but can also consist of deliberately *not* making them (e. g. outwaiting the socially acceptable time frame for the introduction of a new conversation theme). Schegloff and Sacks also pointed out that the conventions typically involve utterance pairs, e. g., one utterance (or silence) to propose the termination, and another utterance in response to accept the proposal. This work, although only concerned with dialogue termination, also illustrates that the social pressure to adhere to these conventions can cause awkward situations. It is quite possible for a dialogue partner to refuse an offering for dialogue termination by, e. g., starting a new strand of conversation. It requires considerable social courage for a dialogue partner to brusquely abort the discourse anyway if another does refuse to cooperate. If we see conversational interaction as a rule-bound game, this difficulty can be ascribed to a reluctance to violate the rules of the game.

(Carlson, 1983) applies the game metaphor for discourse analysis and to determine the semantics of utterances, but he only considers question-answer dialogues. (Lewis, 1979) de-

Game Name	**Joint Goal**	**Goal of Initiator (I)**	**Goal of Responder (R)**
Information Seeking	I knows the information that is sought	I has identified to R the information that is sought	R has provided the information that is sought
Socratic Challenge	I knows whether R can construct particular information i which R plausibly is able to construct based on prior experience	I has identified i to R	R has exhibited R's own knowledge of i
Permission Seeking	Determine whether R gives permission to I to do a particular action a or seek a particular outcome o of action	I has identified to R the action a or outcome o for which permission is sought	R has decided whether R grants permission to I to do a or seek to achieve o

Figure 2.15: Some examples of dialogue macrogames with their joint goals (Mann, 2002)

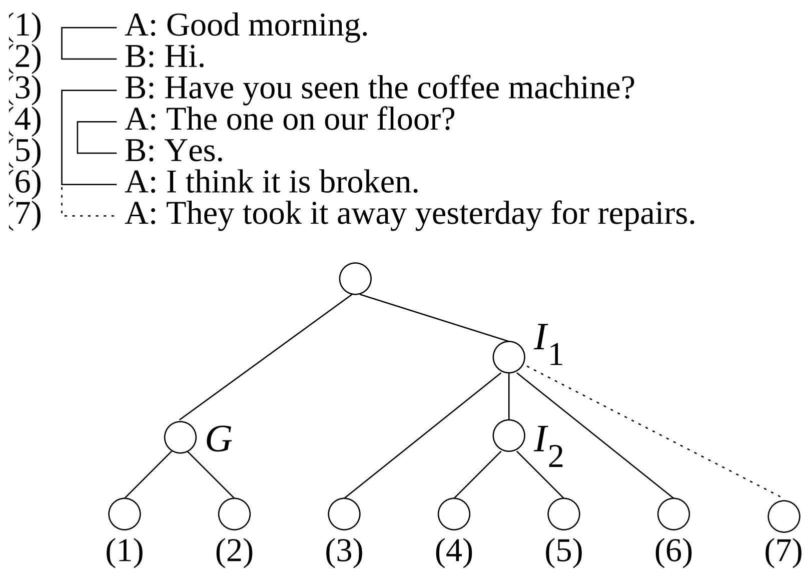
scribes a representation called the "dialogue game board" that is updated with communicative moves during a conversation and thereby keeps a "score" of the common context between the participants. Mann identifies joint goals shared by the participants in dialogues in his investigations on *dialogue macrogames* (e. g., (Mann, 1988, 2002)). Figure 2.15 shows some of the game types he lists, together with the joint and individual goals. The dialogue game metaphor has also been used in various instances for dialogue management and in the analysis of speech corpora for dialogue systems. Examples are the problem-solving in TRAINS (see Chapter 3), the collaborative dialogues in the Edinburgh map task (Kowtko et al., 1991; Carletta et al., 1997), and negotiation dialogues in *Verbmobil* (Alexandersson, 1996).

While dialogues, like competitive games, are goal-directed activities, one characteristic that is harder to pinpoint in them is the notion of *winning* that is central to most classic games; however, some moves may be *better* or more "advantageous" depending on the situation. Dialogue games can be associated with a purpose expressing why they are played, which is usually a joint purpose for all players involved. This can be to share a piece of information, to achieve mutual commitment, etc. In this case, the game is successful if the participants achieve its purpose. However, generally there is no immediate material gain or "winning situation" to a dialogue game[13]. Rather, the point is to avoid to violate the rules—which is socially discouraged—and select from the available moves the one that is suited best for achieving the context and joint purpose (e. g., intuitively, a correct answer should be the "best" reaction to a question in a cooperative context).

Following the terminology used with other games, dialogue games involve a group of participants. The participants take on *roles*. In a game with two participants, the basic roles are the one of the *initiator* that starts the game, and the *responder* role assumed by the other participant. A game consists of a sequence of actions, or *dialogue moves*. The rules of the game specify which moves are permitted at any one point depending on the circumstances, and when the game ends. For the moves, the notion of utterances being acts, namely dialogue acts as described in Section 2.4.1, is adopted.

[13]One can view "socratic dialogue" as an exception; the initiator can "win" a socratic dialogue by forcing the responder to contradict himself or otherwise to accept the initiator's premise.

Dialogue acts are produced by a *speaker* and have a set of intended *addressees* as well as possibly *overhearers*, i. e., participants that were not directly addressed by the utterance, but nevertheless are able to observe the contents of the utterance. Thus, dialogue acts are by their nature *directed* towards an audience; the participants *exchange* dialogue acts. One utterance in a dialogue game is then called a move.



(1) A: Good morning.
(2) B: Hi.
(3) B: Have you seen the coffee machine?
(4) A: The one on our floor?
(5) B: Yes.
(6) A: I think it is broken.
(7) A: They took it away yesterday for repairs.

Game	**Subtree**	**DSP**
G	(1)–(2)	Greeting
I_1	(3)–(7)	Information Seeking (*“where is the coffee machine?”*)
I_2	(4)–(5)	Information Seeking (subgame of I_1; *“which coffee machine?”*)

Figure 2.16: Example of dialogue games and their hierarchical structure

As already mentioned, the exchange of moves is governed by rules; pragmatic social conventions prescribe which sequences of moves are considered acceptable or anomalous. An example of such a rule would be that a question move from A directed towards B should be followed by an answer move from B to A, and that the answer move should actually address the question. Depending on circumstances, other moves are acceptable as well. B could react with a statement that he does not know an answer, or refuse to answer for other reasons. On the other hand, simply ignoring the question is usually not a (polite) option. B could also ask a counter question, in which case a *sub-game* is initiated. The sub-game then is embedded into the original game, which is suspended and continued only after the sub-game has been completed. The resulting structures of sequential moves and hierarchically embedded games can be displayed as a tree structure, as in Figure 2.16. Here, the leafs of the tree correspond to single dialogue moves, and the subtrees to dialogue segment purposes, as shown in the figure table. Note that the end of a game is not always clear-cut in an ongoing conversation: the first information seeking subgame is continued up until turn (7), but it might have been considered already finished after turn (6); the responder belatedly provided some additional information.

Usually, multi-party dialogue is described by means of decomposing it into segments involving two participants at a time. This is often (but not always) adequate. The main shortcoming of this reduction is the failure to take into account that overhearers, or addressees not actively contributing to a segment, at least perceive the utterances and can update their information state accordingly. If they contribute later in the conversation, they can "grab" a role from another participant (e. g., answering instead of the addressee). If their overhearer role is accepted, they can also use rhetoric devices such as anaphoric references to utterances even when they were not addressed in the conversation, as in the example in Figure 2.17, where an overhearer C chooses to take over the responder role from the addressee B.

A ($\rightarrow$ B): *Do you know Peter?*

B ($\rightarrow$ A): *No.*

C ($\rightarrow$ A): *I met him yesterday.*

Figure 2.17: Example: An overhearer takes the role of responder

Dialogue games also present some difficulties. One is that the involved parties need to have shared knowledge about the games used in the conversation; here, discrepancies can lead to confusion. Also, the hearer must, to cooperatively join a game, be able to recognize the type of game, which can be ambiguous if there are several games starting with the same dialogue act type. A related issue is that it may be also ambiguous which dialogue act type a given utterance belongs to. (Pulman, 1996) argues that Bayesian networks may help to identify dialogue act types, however, the mapping may just not be clear-cut in some cases (cf. Section 2.4.1.1). Also, especially in multi-party dialogues, games can occur in an overlapping fashion. These issues are more problematic for a dialogue manager that has to produce a reaction during an ongoing conversation, of which naturally only the previous utterances can help in disambiguation, than in the case of analysis of an already completed dialogue that is known in its entirety.

There have been several formal accounts of dialogue games for different types of dialogues (an overview can be found in, e. g., (Hulstijn, 2000b), Chapter 5.2.1). As an example, we show the approach of (McBurney and Parsons, 2002a,b). There, dialogue games are explicitly called "protocols for interaction". A list of dialogue game components is given, which encompasses

1. *Commencement rules* that define the conditions that must hold for for a dialogue game to begin,
2. *Locution rules* that indicate which utterances (or utterance types) are permitted,
3. *Combination rules* constrain particular locutions to be permitted, not permitted, or obligatory, depending on dialogical context,
4. Moves are associated with *commitments to propositions*. After execution of a move, the initiator is obliged to hold onto its commitments (although it may be possible to back up, this will usually require further action, such as explicitly retracting a claim in an additional game),

5. *Termination rules* define the circumstances under which a game ends. A completed game is called “closed”.

Name	Notation	Meaning
Iteration	G^n	G is repeated n times
Sequencing	$G; H$	G is executed until it is closed, and immediately followed by H
Parallelization	$G \cap H$	G and H are executed simultaneously, until both are closed
Embedding	$G[H\|\Phi]$	H is embedded in G after Φ: G is executed up to some specified point Φ where it is suspended, then immediately H is executed until it is closed, then G is resumed from the point of interruption and executed until it is closed
Testing	$< p >$	Obtains the truth value for the proposition p referring to the world external to the dialogue. This operation might, e. g., consist of a database lookup, or require to conduct a physical experiment to test the proposition

Figure 2.18: Notations for dialogue game combination operations

Composite dialogue games can be constructed from elementary ones using combination operations. A list of combination operations given dialogue games G and H is given in Figure 2.18.[14]

2.5.1.2 The Plan-Based Approach

A second widespread approach assumes that the actions in a dialogue are steps of plans created and pursued by the participants in order to achieve their goals, mainly, to change the mental state of the other participants. Besides an own plan of how to proceed, to understand each other and to be able to react appropriately, the dialogue partners need need to use some sort of *plan recognition*, or a way to communicate their plans explicitly. Some of the informational background for plan-based models were already introduced in Section 2.4.3.1 on the BDI and *SharedPlans* models.

If a dialogue is seen as goal-directed and plan-based behavior, then dialogue can be seen as “a special case of other rational noncommunicative behavior” (Cohen, 1997). Dialogue acts can be defined as operators, manipulating the mental states of the participants, with preconditions and postconditions to assert and retract propositions according to logical inferences over the previous mental state and other context. For a given goal, a possible course of conversational actions can then be devised using standard AI planning techniques, such as HTN planning (cf., e. g., (Nau et al., 1998)).

[14]We think that the last type of operation, testing, is not strictly required, but can also be replaced by embedded sub-dialogues, such as database request-response interactions.

BOOK-FLIGHT(A, C, F):

Constraints: agent(A) $\wedge$ flight(F) $\wedge$ client(C)

Precondition: know(A, depart-date(F)) $\wedge$ has-seats(F) $\wedge$ want(C,BOOK-FLIGHT(A, C, F)) ...

Effect: flight-booked(A, C, F)

Body: make-reservation(A, F, C)

INFORM(S, H, P):

Constraints: speaker(S) $\wedge$ hearer(H) $\wedge$ proposition(P)

Precondition: know(S, P) $\wedge$ want(S, INFORM(S, H, P))

Effect: know(H, P)

Body: believe(H, want(S, know(H, P)))

Figure 2.19: Some *STRIPS*-like action schemata for flight booking

Act axiomization can use, e. g., parameterized *action schemas* derived from operators for the *STRIPS* planner (Fikes and Nilsson, 1971). An action schema has *preconditions* that must hold for it to be applicable, *effects* (also called *postconditions*) that are introduced when the schema is performed, and a *body* that specifies subactions to be performed, or subgoals to be fulfilled when the schema is executed. Figure 2.19 shows an two abridged action schemata that are part of an example from (Jurafsky and Martin, 2000). The first schema states that to book a flight for a client C, the agent A must know (among other things) the departure date of the flight. If C is aware of this requirement, it can instantiate the INFORM action schema and produce an utterance to this effect. Otherwise, A could try to elicit the INFORM action by producing an instance of an INFO-REQUEST schema (not shown in the figure), to explicitly communicate to C that A needs this information.

One could question whether a planning stance actually matches the approach of human dialogue partners. There is no doubt that dialogue constitutes is a cooperative and often goal-directed activity, and that the interlocutors generally have some idea of how to arrive at goals and sub-goals. However, the nature of dialogical interaction itself exhibits some features that speak against treating it as a classical planning problem. Intuitively, humans interlocutors seldom plan ahead in all detail how a conversation will or should proceed. There are some exceptions in rigidly formalized conversations (such as an officer asking someone to answer a series of questions for a questionnaire) or carefully premeditated arguments where one party has “hindsight” about the contributions of others (such as a socratic dialogue). However, even in such cases, there will usually be exchanges that are not part of the “plan”. Instead, the contributions during dialogue are chosen and formulated opportunistically, one turn at a time, and in real time. Also, the assumption of perfect information—that a planning agent knows the full state of the world it plans in—is not a given in common dialogues, and no attempt is made to establish it initially; instead, the participants adapt their course of action to information that is discovered during the interaction. However, humans *do* employ planning on the level of a task they are trying to achieve.

Plan-based dialogue processing presents difficulties with respect to the interlocutors being required to interpret each other’s utterances, that also may be ambiguous, and which requires them to employ *plan recognition*. Similar to dialogue games, for a plan to be recognizable, both parties need to have shared knowledge about it. The approach is generally considered to be powerful, but also to require more complex domain modeling (Allen et al., 2001a), as well as computational effort if the plans are computed dynamically during the interaction: in the worst case, planning and plan-recognition can be combinatorially intractable or undecidable (Cohen, 1997). To reduce the plan-creation effort, some systems retain libraries of generic, prefabricated plans as a special kind of belief in the knowledge base (e. g., (Georgeff et al., 1999; Larsson et al., 2000)). This kind of plans are called *recipes* and are especially helpful for schematic situations that occur frequently. Models with different levels of goals and plans have also been proposed, e. g., a tripartite plan-based dialogue model in (Lambert and Carberry, 1991) that has separate plan libraries for domain, problem-solving, and discourse issues.

2.5.1.3 Turn Management

Spoken conversation overwhelmingly proceeds in *turns* where one party—the one "having the turn"—speaks at a time, and human speakers have means to finely coordinate the allocation of turns (Sacks et al., 1974). Each turn also provides a natural interval for information state update. For coordinated dialogue, the participants need to take heed of time constraints. For one thing, available "half-duplex" modalities such as speech, (i. e., ones that are unreliable if they are used in a simultaneous fashion) have to be assigned to one person at a time; others, e. g., gestures, can be used and understood in parallel by all participants. There needs to be some management of the available modality resources (also called *communication channels*) over time to avoid conflicts and misunderstandings. The right to talk can be asserted verbally (e. g., by explicit assignment, as in *"What do you think, Robert?"*, or by nonverbal means such as using gestures and eye gazes. Humans are very proficient in recognizing the points in conversation where the turn assignment can change, the so-called *transition relevance points* (TRPs). In human-human dialogues, most of the time, there is no or very little discernible overlap or gaps between turns (Sacks et al., 1974).

The need for turn-taking management is largely eliminated if a system uses *single initiative*, i. e., the interaction is driven by one of the participants, the other only reacts. Pure *system initiative* is prevalent in many simpler system-user interactions for more linear task-based applications. In this case, the system issues a *prompt*, often in form of a question, to indicate a turn yield and the expectation of a user contribution. For example, telephone based service hotlines often feature dialogues like the following:

SYSTEM: *"Good morning, welcome to <company name>. Please answer loud and clearly. Do you want to talk about your Internet service contract, technical problems or other issues? Please say 'contract', 'technical' or 'other' . . . "*

A prompt can, as exemplified by the second sentence, also constrain possible utterances by the user by already offering a limited set of options to choose from. This is useful if, e. g., the system can only handle a small vocabulary. System-initiative dialogue can be tedious if the user knows what she wants but needs to conform to the system's terms to get there, maybe involving several fixed steps in between that could be avoided. Interfaces that employ pure *user initiative* are very rare; since they impose on inexperienced users the problem to find out what exactly is possible in the interaction.

```
while conversation is not finished
  if system has obligations
     address obligations
  else if system has turn
     if system has intended conversation acts
       call generator to produce NL utterances
     else if some material is ungrounded
       address grounding situation
     else if some proposal is not accepted
       consider proposals
     else if high−level goals are unsatisfied
       address goals
     else release turn or attempt to end conversation
  else if no one has turn
     take turn
  else if longer pause
     take turn
```

Figure 2.20: A discourse actor algorithm (Traum and Allen, 1994)

More flexible and natural are *mixed-initiative* systems where each dialogue partner can take the initiative. (Traum and Allen, 1994) gives a *discourse actor algorithm* for mixed-initiative plan-based systems, shown in Figure 2.20 (the update of the conversational status with newly perceived conversation acts progresses asynchronously to this algorithm in a parallel thread). The algorithm is a continuous loop that in each iteration will decide whether to take an action, depending on the current context. Taking and releasing turns can happen implicitly (e. g., by pauses), but nevertheless are actions in their own right. (Traum and Hinkelman, 1992) introduced a class of *turn-taking acts* containing acts *take-turn*, *keep-turn*, and *release-turn* (which has a subvariant *assign-turn* if the releasing party explicitly selects another as the next speaker). In an environment involving more than two interlocutors, there is an additional element of multiple participants competing for the next turn. Turn-taking acts can, beyond their "protocol" function, also take shape in the system presentation in various forms; e. g., a visual prompt indication for the user, or, in the case of virtual characters, explicit gestures and gazes akin to turn-taking signals between human interlocutors.

(1)	USER:	*"Book a ticket for Thursday."*
(2)	USER:	[before System responds (during processing)] *"No, I mean Friday."*
(3)	SYSTEM:	*"Here is your reservation number for . . . "*
(4)	USER:	*"Actually, make that two tickets."*

Figure 2.21: Examples for "barge-before" and "barge-in"

In this context it is also significant what happens during interruptions as in the example of Figure 2.21. It can happen that a participant interrupts during another's turn (*barge-in*, turn (4)), or the user makes an utterance during the move preparation phase of a dialogue system, but before the start of the utterance realization (*barge-before*, turn (2)) (Beringer et al., 2001).[15] An adequate reaction to such interruptions can require (a) aborting an utterance during realization, as in turn (3), and (b) re-evaluating or reconstructing adopted plans. A plan-based system can check for interruptions during and after the move planning phase and before actual realization, but there still remains the delay caused by the realization steps after the action manager, such as text generation and speech synthesis.

To proactively reduce the incidence of barge-in and barge-before, the system can provide visual or other feedback cues to the user when the virtual characters actually expect her to say something, or when the characters are themselves busy preparing to make an utterance. The emphasis of such a mechanism depends on the circumstances. In a task-oriented dialogue system, where miscommunications are potentially disruptive and frustrating, a more bold or even exaggerated display of attention may be appropriate (e. g., in the *SmartKom* system, the system character *Smartakus* caps its ears in an obvious and even comical gesture), while in a narrative context, more subtle hints like gazes are probably advisable, to avoid disturbing the atmosphere.

2.5.1.4 Grounding

During a conversation, the participants build up a *common ground*, i. e., shared assumptions about what was said. Adding to the common ground is called *grounding*. (Clark and Schaefer, 1989) list as main types "evidence" for grounding: *continued attention*, indicating an utterance was accepted and there is "nothing wrong" with it, the *initiation of a next relevant turn*, explicit (and often overlapping) *acknowledgement*, e. g., by short cue words or an evaluation like *"that's great"*, the demonstration of what the recipient has understood by *completion* or reformulating the utterance in a *display* of the content of the original utterance. If a contribution was *not* or just partly understood, the receiver can answer with a request for repair pointing out the problem, either requesting a restatement of the full utterance (*"I beg your pardon?"*) or the part in question ("when *did you say you wanted to go to Amsterdam?"*).

Type	**Function**
Initiate	Initial utterance of a discourse unit
Continue	Continuation of a previous act by the same speaker
Acknowledge	Shows understanding of the previous utterance
Repair	Changes the content of the current discourse unit
ReqRepair	Request for repair by the other party
ReqAck	Attempt to get the other agent to acknowledge the previous utterance
Cancel	Closes off the current discourse unit as ungrounded

Figure 2.22: Grounding acts

For explicit grounding and requests for grounding, (Traum and Hinkelman, 1992) introduced a set of *grounding acts* for discourse units (see Figure 2.22), whereby a discourse unit is an initial utterance—an attempt to realize a core speech act—and *"as many subsequent utterances by each party as are needed to make the act mutually understood"*. Since utterances are often already understood before they are completed, grounding actions often uses non-verbal

[15]This does not include short feedback interjections like *"uh-huh"* or *"yes"*, which are used for grounding.

communication channels, like nodding and gazing, instead of (short) utterances, so as not to interrupt the speaker.[16]

Apart from producing the external acts to confirm or request grounding to the other speakers, the participants themselves have to change their information state to integrate a perceived dialogue act after it has been accepted as part of the common ground. This can involve merging the new information with elements gathered in previous turns (or assumed "a priori" by the dialogue agent), or possibly even resolving incompatibilities by dropping knowledge items. Chapter 4, Section 4.4.3 describes a way to achieve this.

2.5.2 Multi-Party Dialogue

Much remains the same when going from dialogues with two participants to multi-party dialogues (conversations), but some issues gain additional complexity. It also plays a role for a dialogue manager on which "side" new participants join in: are there additional human users, or does the system manage several virtual characters? How can different human users be told apart? And how many aspects must be modeled separately for each character by the system?

In this section, we take a look at some important aspects of multi-party interactions: the variations in the roles of the participants, speaker and addressee identification, and how the process of turn-taking changes.

(1) **contact**

(2) **attention**

(3) **conversation**

- (i) participants
- (ii) initiative
- (iii) grounding
- (iv) topic
- (v) rhetorical
- (vi) turn

(4) **social commitments (obligations)**

(5) **negotiation**

Figure 2.23: Multi-party, multi-conversation dialogue layers (Traum and Rickel, 2002)

(Traum and Rickel, 2002) identify the topics and sub-topics shown in Figure 2.23 regarding multi-party, multi-conversation dialogues and propose that their processing should be arranged in layers (one per item), as each item depends on the former. While the topics also occur in two-party dialogues, they are more complex in the multi-party case. The model introduced in this thesis is mainly concerned with the items (3)(i)-(iv) and (4). For (1), it

[16]In situations where it is exceptionally important to be sure about common ground, such as radio traffic in a rescue situation, explicit confirmations are much more common. An example is the excerpt from emergency communications in the Apollo 13 mission cited by (Mann, 2002).

is assumed that all participants are in contact and available for communication (however, a perception filtering mechanism (cf. Section 5.4.2) can result in participants overhearing an utterance). Behavior regarding (2), such as eye-gazing is implemented by the *FADE* module in the systems described. In addition, the participants have to recognize who is making an utterance, and whether they are addressed. This is also closely tied to turn-taking (3)(vi), which is handled by the behavior generator and *FADE* together (see Section 7.2.4). Rhetorical connections between the dialogue acts and obligations are captured by the possible moves in dialogue games and possibly additional constraints on the activities they occur in, as will be shown in Chapter 5. Finally, level (5)—negotiation of goals and plans—is not covered by the model.

The following sections take a closer look at the issues of speaker and addressee recognition, the participant roles and the management of turns and conversation threads in multi-party conversations. For this, we largely follow the analysis in (Traum, 2004).

2.5.2.1 Speaker and Addressee Recognition

In multi-party dialogues it is necessary to determine the speaker and/or the addressee(s) of utterances, especially if speech recognition is involved, and multiple participants may use the same modality. Also, an utterance can be addressed at a group instead of a single participants. For systems where speaker recognition is a practical concern rather than a research interest, communication messages sent between agents can include an explicit speaker identification, and the hardware setup can be used to distinguish different human speakers.[17] Intended addressee(s) can be made clear explicitly or implicitly in the content of the utterance itself, e. g., by using a vocative. In the case of computational agents participating in the dialogue, the multi-party situation first *introduces* the necessity to perform speaker and addressee identification, where in a dyadic interaction, it is clear that "I must be addressed when the other one is speaking" and vice versa.

```
if utterance explicitly specifies addressee
    addressee ← specified addressee
else if speaker is the same as speaker of immediately previous utterance
    addressee ← previous addressee
else if previous speaker is different from current speaker
    addressee ← previous speaker
else if unique other conversational participant
    addressee ← participant
else
    addressee unknown
```

Figure 2.24: Addressee identification algorithm for multi-party interaction (Traum, 2004)

Figure 2.24 shows the algorithm used for addressee recognition in the *MRE* system (Traum, 2004). For the general case, this is however only a heuristic, and not appropriate in all cases. For example, there are ways to specify the addressee of an utterance that do not depend on the utterance itself, but, e. g., the general conversational and social context (e. g., if the

[17]For example, in *VirtualHuman*, human users can be distinguished because each uses a dedicated microphone.

speaker gives an order, the addressee will likely be a subordinate rather than a superior), situational context (e. g., spatial proximity), task role, or via communication on additional modality channels (e. g., using gazes and gestures).

2.5.2.2 Participant Roles

Two-party dialogues exhibit the fundamental roles of *speaker* and *addressee*, which can be switched according to who currently has the initiative. In multi-party interactions, the situation is more complex. The addressee (whether it is a single participant or a group) is no longer the only listener, but is joined by others taking *overhearer* roles of various categories. An overhearer is a bystanding dialogue participant that is not directly addressed by a contribution, but perceives and possibly reacts to it. One can further differentiate whether the overhearing is intended/unintended, voluntary/involuntary, or whether the overhearer is in-context/out-of-context. Overhearers can be implicitly included in or the conversation, and cause the speaker to adapt what she says in other ways. During the interaction, the speaker and hearer roles can shift between being an active participant, an (active) listener, or an entirely passive bystander.

Other dimensions are defined by the *social* and/or *task* roles. Depending on the subject matter, social roles can determine task roles or vice versa. These roles can influence the frequency and duration of contributions and the determination with which a participant holds or grabs the turn (Rumpler, 2007). They also establish relations such as, e. g., the authority to lead the conversation, or issue commands to a subordinate (Traum, 2004). In some situations, they can also help to disambiguate the addressees of an utterance.

2.5.2.3 Turn and Conversation Thread Management

When multiple participants are involved, turn management is more complex than in the two-party case. For example, in the two-party case, the turn-taking act *release-turn* is (almost) identical to *assign-turn*, since there is only one possible other speaker. It is also possible that multiple parallel threads of conversation, with different participant sets, are active at the same time. If these threads are conducted on the same exclusive communication channels (e. g., speech), turns need not only be negotiated between participants in the same conversation, but also between conversations, to avoid the understanding problems that occur with overlapping speech. It has also been noted (Cohen et al., 2002) that in multimodal multi-party dialogues, there are situations where several participant cooperate across modalities, e. g., one participant is speaking, and the other supplies an illustrative gesture; both actions could be interpreted as a combined, but single multimodal contribution.

It is possible in two-party dialogues that more than one conversation thread is pursued in parallel, but it is much more common in multi-party conversations. Traum notes that in this case, *“it can be tricky to determine which thread a particular utterance belongs to”* (Traum, 2004, p. 6), even if the topical structure is taken into account. The model presented in Chapter 5 allows parallel and nested sub-conversations with different participants; however, ambiguities in assigning contributions to particular activities have to be resolved on the task level.

2.5.3 Dialogue System Architecture

A full-fledged dialogue system has to address a variety of tasks, which are quite different in nature. When investigating dialogue management only, some restrictions can be imposed to be able to ignore some factors perceived to have lesser impact on the interaction *per se*, such as requiring that the conversation is typed, as in, e. g., (Jönsson, 1993), or the utterances are selected from a set of options (Rich and Sidner, 1998). This simplifies the task and makes some areas of processing obsolete, possibly allowing the researcher to concentrate on phenomena she actually wants to study.

Such restrictions can, however, also move a system away from capturing "natural" spoken interaction, which is considerably different from, e. g., typed conversations. Important issues arising of human-human conversation may be overlooked, even though they *are* relevant to conversation management. Some phenomena that are encountered in spoken conversation do less frequently occur in written form (e. g., false starts, noise) or are not perceived (e. g., hesitations, accents). If the input is typed, the characteristic errors—typos—are quite different in nature to misunderstandings in spoken conversation: the string *"good monring"* is a typical typo, while *"good mourn ink"* would be a plausible speech recognition error, but is unlikely to occur in typed input. Lastly, a reduced setup may also suffer from taking away the modality dimensions in human interactions, like the ability to use gestures, prosody, and facial expression.

We aim to design a conversation manager for as natural a system as possible, although it can and has been argued that it is a good strategy to make good conversation managers for such restricted systems first, and then expanding to the more general case (Allen et al., 1996). Additionally, we consider systems using multiple input and output modalities in addition to speech.

2.5.3.1 Order of Processing

Figure 2.25 shows a way to put the different tasks necessary to do end-to-end multimodal dialogue processing in a sequential order, from recognizing input to synthesizing output. Note that the depiction does not necessarily include all tasks of such a system. It does not, e. g., incorporate tasks related to application logics or knowledge representation. This fragmentation of the dialogue management task corresponds with different roles in speech production and understanding in human conversation.

(Clark, 1996) points out that tasks appear symmetrically for the speaker and hearer of an utterance, i. e., (a) vocalizing and attending utterances (multimodal analyzers and renderers – TTS, player), (b) formulating and identifying the message (multimodal interpretation, fusion, and DiM on one hand, fission and multimodal generators on the other), and (c) conceptualizing and comprehending the content (action manager). This has led to some efforts to let modules handle both directions.[18] Action planning, on the other hand, always entails to both analyze input from the user and generate output for the user.

[18]For example, the *SPIN* language interpretation component (Engel, 2002) was used as a generator, *NIPS*, in the *OMDIP* system described in Chapter 7.

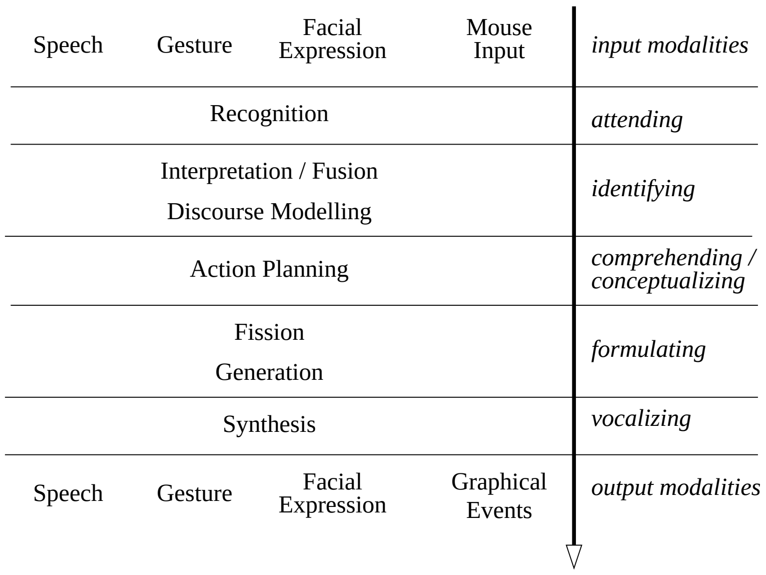


Figure 2.25: Processing sequence from multimodal analysis to multimodal generation (adapted from (Clark, 1996))

2.5.3.2 Modularization and Communication

Early dialogue systems, e. g., *TRAINS*, used a *monolithic architecture*, essentially collapsing the different subtasks in the processing sequence in one program. However, this approach showed hard to construct, debug, and modify for new tasks and domains (Allen et al., 2000b). This led to a practice of encapsulating the subtasks in dedicated *modules* that communicate with each other. There are different approaches to realize this communication, as shown in Figure 2.26. In a *pipelined architecture*, the data flow between modules is sequentialized in an order more or less equivalent to moving top-down in Figure 2.25. The *TRAINS* system is (mostly) unidirectional in that sense (Ferguson and Allen, 1998). If sequential ordering is imposed strictly, it prevents some useful features. Modules early in the processing chain could often profit from the availability of results of modules at a later stage.

Why it is useful to allow communication to go "backwards" against the canonical processing sequence is exemplified by the phenomenon that human dialogue partners prefer to re-use terms that other participants (or they themselves) have already used earlier in the conversation over the introduction of different terms that may be semantically equivalent. This can be explained as an instance of Grice's maxim "avoid obscurity of expression" (Grice, 1975), since it eliminates the cognitive load of establishing the equivalence. Therefore it can be beneficial to have a (two-way) connection between the speech generation and the speech recognition component to coordinate input and output. Also, new lexical terms can arise during the interaction, e. g., when expressions that are not commonly present in a lexicon are newly introduced into the discourse as the result of database lookups, as in the *SmartKom* EPG application (Löckelt et al., 2002) and in *OMDIP* (cf. Section 7.3.1).

Although it is also possible to connect all modules that need to communicate by direct channels, most state-of-the-art systems today use modules that are independent, concurrent agents

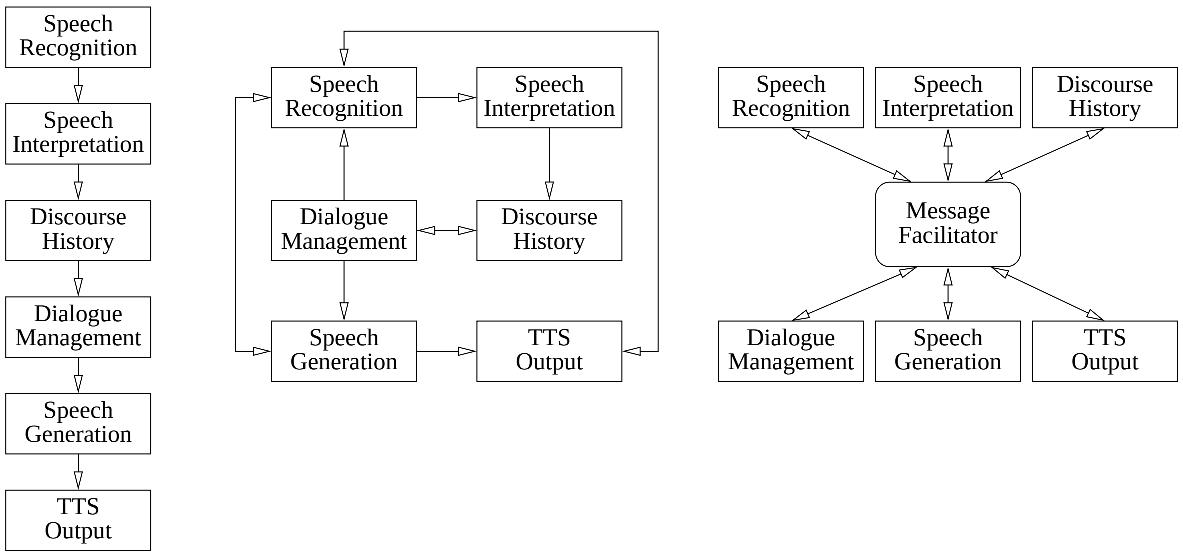


Figure 2.26: Sequential, direct and facilitator-mediated communication between modules

that can communicate freely with all other agents by message passing. This follows the reference architecture for multimodal systems given in (Maybury and Wahlster, 1998), shown in Figure 2.27. A way to provide free message passing in a straightforward way is using a *blackboard architecture*. It features a *facilitator module* managing a shared, central data pool (the *blackboard*) where modules can post typed messages. Other modules communicate with the facilitator to either read messages directly off the blackboard, or they can subscribe messages of certain types that the facilitator will deliver to them. *Multi-blackboard* systems use several blackboards in parallel. One advantage of blackboard systems is that the modules are *decoupled* (they are only communicating directly with the facilitator) and can organize their input and output solely in terms of message types to send or receive. Subscribers can be added or removed without affecting the posting modules. An example of a multi-blackboard system is *MultiPlatform* that was used in the *SmartKom* system (Herzog et al., 2003). *SmartWeb* uses a Java-based *hub-and-spoke* architecture (*IHUB*) where a central module decides from a set of rules where messages should go (Reithinger and Sonntag, 2005).

A multi-agent approach also has some drawbacks. First, there are software engineering issues that arise from an architecture with concurrent modules, such as possible deadlocks and timing problems. Second, in the absence of clearly defined processing steps, it can be difficult to decide the beginning and end of "turns", and problems can arise with race conditions and deadlocks. These and related issues can make concurrent multi-agent systems somewhat harder to analyze and debug.

2.5.3.3 Module Types

The dialogue management framework described in this thesis and its predecessor, the *SmartKom* action planner, work in different systems comprising all the types of modules shown in Figure 2.28; they are directly connected and involved with the modules shown

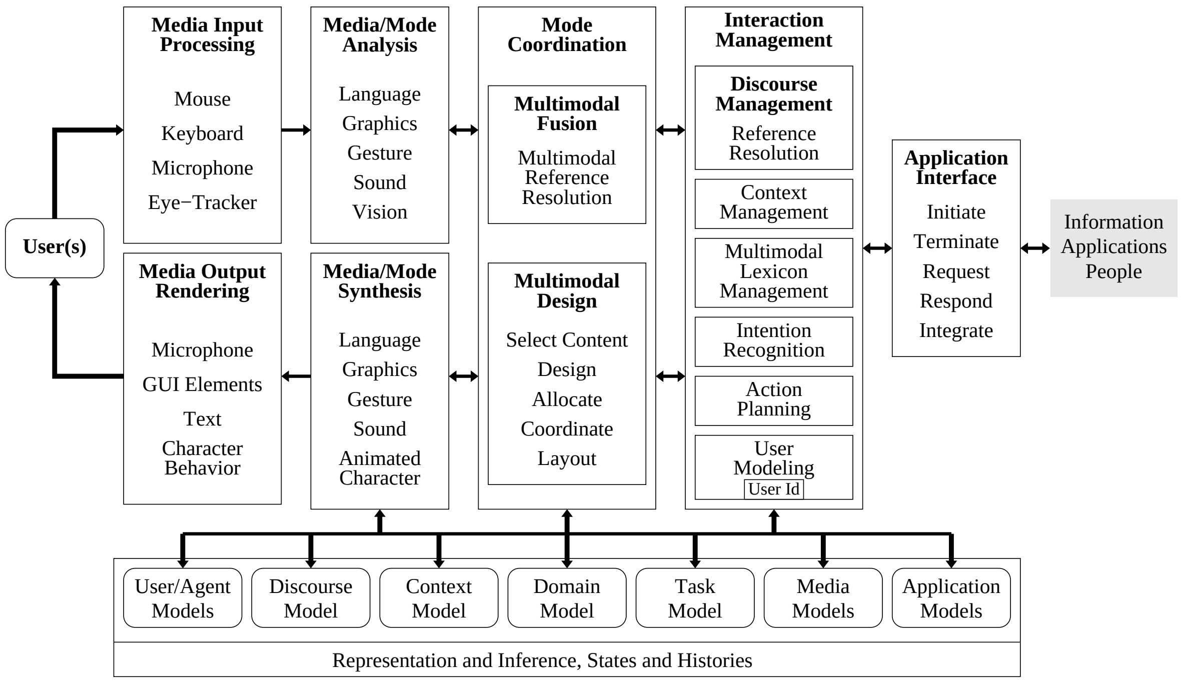


Figure 2.27: Architecture for multimodal systems (Maybury and Wahlster, 1998)

in bold. The most essential ones are:

- **Discourse modeler**

  The output of the discourse modeler is the main input for the dialogue manager. It is in terms of dialogue acts that already incorporate the information gathered from fusion of different modalities and resolution of references against the discourse history. In some systems, discourse modeling is integrated with dialogue management. As stated in Section 2.4.2.2, we found it advantageous to use two separate modules for these tasks, however, in all instances the same knowledge representation formalism was shared between both modules.

- **Function modeler** and **multiple application frontends**

  A function modeler is a module located between the dialogue manager and the application frontends that provides an abstraction layer so that the latter does not have to implement all idiosyncrasies of the application protocols. It is particularly useful in setups where the dialogue manager has to interact with many heterogeneous applications. The framework we will describe assumes that it can communicate with a function modeler in form of dialogue acts, and that the application frontend interaction protocols can be specified in terms of dialogue games.

- **Presentation planner** resp. **presentation module** and **multimodal generators**

  To let the output of the system or the virtual characters be rendered, the dialogue manager sends it to a presentation planner, or, if no such module is present, directly to

Module	Purpose	*D*
affect engine	computing affective state of system characters based on the interaction and their personality traits, influencing mood-dependent idle behavior	$\leftrightarrow$
discourse modeler	storing and organizing discourse history, reference resolution	$\leftrightarrow$
dynamic help	monitoring the dialogue and offering context-dependent help or other support to the user	$\leftrightarrow$
dynamic lexicon	parameterizing speech recognition with context-dependent lexicon entries in interaction with the dialogue manager	$\rightarrow$
function modeler	providing a unified interface to multiple applications and application groups by providing an abstracted access layer	$\leftrightarrow$
generators	converting symbolic output representation into a sequence of syntactic elements that can be rendered by the presentation module	$\rightarrow$
intention recognition	selecting the most probable input hypothesis based on context and other criteria	$\leftarrow$
multimodal fusion	combining and integrating acts in different modalities, resolving cross-modal references	$\leftarrow$
narrative engine	controlling characters externally by manipulating their goals. This can be combined with processing of goal state feedback to adapt the story	$\leftrightarrow$
planner frontend	uses an external planner program to solve planning problems for the dialogue manager	$\leftrightarrow$
presentation module	realizes the system output using 2D or dynamic 3D rendering	$\leftrightarrow$
presentation planner	planning layout and (partial) modality selection	$\rightarrow$
prosodic analysis	influences the user model	
recognizers	getting and converting device input signals for analyzers	
synthesizers	producing output presentations, (e. g., TTS)	
analyzers	interpreting analyzer input and converting it to a symbolic representation	
application frontends	providing adapters translating the internal message format for applications	
other modules		

Figure 2.28: List of module types. The rightmost column indicates the direction(s) of communication with the dialogue manager (right=outgoing, left=incoming, or both)

a presentation module. Again, the units of communication are dialogue acts containing a semantic, symbolic encoding of the content. In cases where the surface output form is not contained as "canned text" in the semantic representation, it is therefore necessary to let it be processed by a generator module to obtain a surface form (Reiter and Dale, 1999). Deep generation and canned text can also be used together in the same system (cf. Section 7.2).

- **Narrative Engine**

  In narrative systems, the story can be controlled by a narrative engine module. It offers the possibility to let an author designer develop the storyline separately from low-level dialogue issues. The module can directly influence the behavior of virtual characters by setting their goals and receiving feedback from the dialogue manager about their execution state.

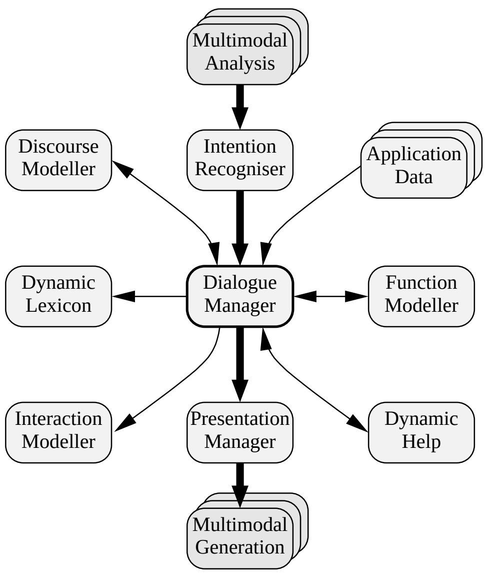


Figure 2.29: Constellation of modules in direct communication with the dialogue manager (example *SmartKom*)

Depending on the focus of a given system, it can also be advantageous to integrate other modules closely with the dialogue manager. For example, dynamic help, recognition and analyzer modules can benefit from generated expectations about future input (see section 6.3.2). As an example, Figure 2.29 shows the modules that are directly connected with the dialogue manager in *SmartKom*.

2.5.4 Discussion

We aim for a model that can be used to implement a practical system. This puts at a disadvantage theoretical tools that, while being possibly more expressive or powerful than others, are too complex and have high demands with respect to memory usage, computational cost, and necessary effort in modeling non-trivial domains. On the other hand, we also want to avoid overly simplistic approaches that do not allow complex dialogues, or just deliver reasonable performance when limited to trivial domains.

Our approach will strive to offer both possibilities: The basic conversation model will not automatically require extensive logical inference and planning capabilities, but leave open the option to include sophisticated reasoning mechanisms if they are required by the application. We also recognize the advantages of approaches that can benefit from accepted standards and widespread tools, as well as ones that allow to re-use existing work. This especially holds true for knowledge representation, for which the Semantic Web effort has introduced ontological modeling languages that are now increasingly adopted by the community to model and share application domains.

2.6 Summary

This chapter treated the requirements for the two types of dialogue systems we are concerned with, which are task-oriented systems and interactive narratives, and the basic concepts necessary to understand the operation of a dialogue manager.

We first examined the purposes of both dialogue system types, and which criteria can be applied to determine whether it is successful in addressing the purpose. We then introduced the *Interaction Triangle* which shows the relationships created by the system between human users, the designer of the system and its content, and the virtual characters acting as system avatars. The section on dialogue modeling was concerned with dialogue acts, the basic units of dialogue, how they fit into larger dialogue modeling structures, and the role of information state for a dialogue system. In this context, we presented the BDI model, *SharedPlans*, as well as the information state representations in *TRINDI* and *SmartKom*. We then described a number of issues that multimodality adds to the dialogue management task. The fourth section dealt with the basic concepts of dialogue management. Dialogue games and the plan-based approach as means to model it were introduced, together with issues related to turn taking and grounding contributions for all participants. Multi-party dialogue, again, adds some further phenomena, like variable and shifting participant roles and changes in interaction and turn management. Finally, we described how systems that do dialogue management are organized and modularized.

Chapter 3

Related Systems

3.1 Introduction

The field of dialogue research as seen a wide variety of dialogue systems. An overview in the scope of this thesis cannot be exhaustive, therefore we are aiming to cover different approaches and areas of application, and give an overview of the most influential approaches and implementations that relate to our field of study. Strictly speaking, any interaction with a computer could be viewed as a (possibly formalized) dialogue with the machine; however, we start our overview with systems that could process spoken input in natural language, and focus on multimodal systems. We also try to avoid too much overlap in describing systems that are too similar in nature, and instead pick a selection of systems, from which each has to offer relevant distinctive features or characteristics. In the two main sections of the chapter, we cover examples for task-oriented systems and interactive narratives, respectively; however, the distinction is not always clear-cut.

3.2 Task-Oriented Systems

In the following, we describe the *TRAINS* and *TRIPS* systems, which are examples of projects that use a logic domain description and AI planning techniques. The *COLLAGEN* project features multimodal interaction and uses the *SharedPlans* theory to model collaborative action. *RavenClaw* is an instance of a general framework for task-oriented dialogue systems that separates task and domain knowledge. *TrindiKit* is an influential toolbox approach based on the notion of "information state" that has been used in several systems. *WITAS* emphasizes real-time control, joint activities, and a dialogue model using a dialogue tree that is similar to our approach. *MATCH* is a multimodal system that that is entirely finite-state based and runs on a mobile device. *QuickSet* has been developed as a research tool to examine multimodal interactions. *SmartKom* is a large, multimodal, multi-application system with ontological domain modeling and multiple coordinated modules whose action planning approach is a direct precursor to our framework. *SmartWeb* offers information-seeking dialogues in the Semantic Web using advanced ontological domain modeling; its infrastructure has been used along with the framework described in this thesis to implement one of the use case systems described in Chapter 7.

3.2.1 *TRAINS* and *TRIPS*

TRAINS The *TRAINS* system is a long-term research project that has gone through several versions from *TRAINS*-91 to *TRAINS*-96 (Allen et al., 1995; Ferguson et al., 1996).[1] An untrained human user is assisted by the system in solving routing problems in a simplified train transportation domain of five cities. One such problem could be, e. g., to send *"a boxcar of oranges to Bath by 8 am"*. The system shows a map with the locations of destination cities, freight train engines, and possible routes, as shown on the left side of figure 3.1. When the user makes suggestions of what actions to take, the system calculates expected route times and scheduling conflicts, and suggests solutions to problems that may arise.

The discourse manager of *TRAINS* is composed of several submodules for context representation, reference disambiguation and an actualization component holding models of the system "self" and the display (Traum, 1993). The system uses domain-specific reasoning instead of full planning techniques to overcome performance issues. The system architecture of *TRAINS*, in terms of modules and communication channels, uses a fixed rather than a more versatile, e. g., blackboard-driven, setup by deliberate design decision, to trade-off flexibility for simplicity (Allen et al., 1995).

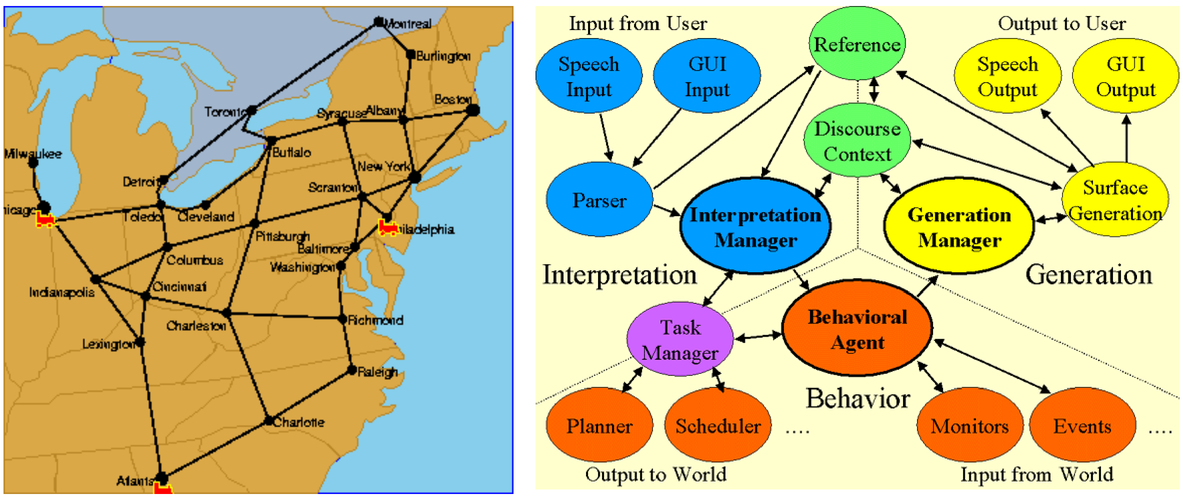


Figure 3.1: (Left) a later *TRAINS* domain, (right) the *TRIPS* architecture (from the project website)

Early versions of *TRAINS* were implemented to demonstrate that constructing a system that does robust processing of spontaneous dialogue with non-expert users about a narrow domain in real time is possible in practice. The system was also intended as a research platform, and subsequent versions were more robust and added functionality. *TRAINS*-96 breaks up the previously monolithic system into a modular design. Communication is changed from using a custom representation for episodic logic (Hwang and Schubert, 1993) to the *KQML* agent communication language (Finin et al., 1994). Several versions of *TRAINS* have been evaluated for time to task completion and length of solutions, e. g., in (Allen et al., 1996; Sikorski and Allen, 1997). Also, a set of 98 task-oriented dialogues involving 34 speakers has

[1]Project website: *http://www.cs.rochester.edu/research/trains*

been collected during the project and made available as the "*TRAINS* corpus" (Gross et al., 1993). This corpus has been used as a reference in numerous other research projects.

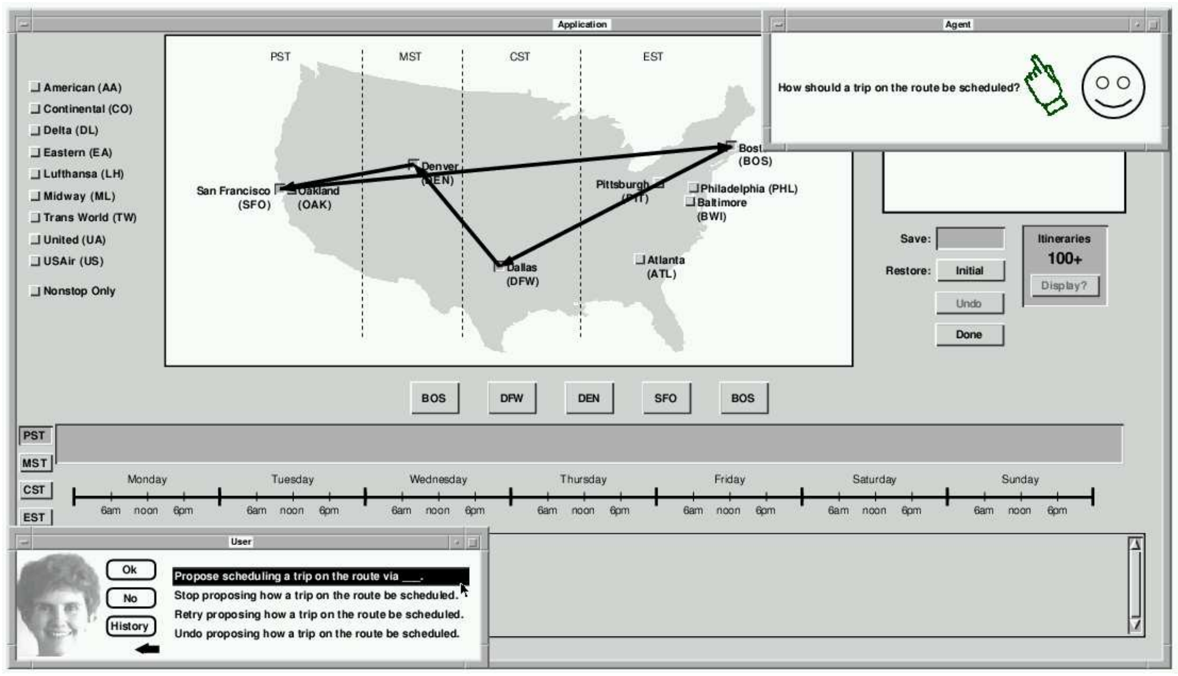
TRIPS The successor of *TRAINS*, *TRIPS* (Allen et al., 2001a; Ferguson and Allen, 1998; Allen et al., 2000b) has further emphasis of being an end-to-end system including all components necessary to do research about concrete collaborative planning problems, constituting an *"integrated AI system"* (Ferguson and Allen, 1998). *TRIPS* is not a completely new system, but builds on and reuses many of the components from *TRAINS*. It employs collaborative planning assistants and uses a hub and spoke architecture to convey messages between the agents in the KQML language (Finin et al., 1994). It was used to implement multiple different domains, e. g., scenarios coordinating rescue vehicles during emergency situations (*PACIFICA* and *TRIPS-911*) and controlling virtual robots doing exploration (*Underwater Survey*). Overall, the scenarios and the problem solutions are more complex than in *TRAINS*. *TRIPS* features incremental interpretation and generation, separation of task and domain reasoning from discourse reasoning.

As envisaged, *TRAINS* and *TRIPS* pioneered as complete, usable, and extensively evaluated end-to-end systems for task-oriented dialogue in changing task settings. Most current systems use a similar partition of work into modules and comparable module communication techniques. This work uses a set of dialogue acts that draws on work on the *TRAINS* corpus (Core and Allen, 1997; Traum and Hinkelman, 1992). Both projects had foundational influence on dialogue management, however, both also exemplified the problems incurred by the modeling of more complex domains (Allen et al., 2000a).

3.2.2 *COLLAGEN*

The *COLLAGEN* system is intended as a *collaboration manager* to maintain the flow and coherence of collaboration between a system agent not part of *COLLAGEN* proper, and a user, jointly working on the same task (Rich and Sidner, 1998). The example scenario involves an application to plan air travel schedules. The user can interact with the application and with an agent. The agent is a black box module: it is not specified what mechanism it uses for deciding what contributions to make to the discourse, or what actions to initiate. The developer of an agent for *COLLAGEN* needs to specify a task model, but as Rich and Sidner emphasize, this model only relates to communication and collaboration with the user, and leaves aside the decision making of the agent, which can be implemented in other ways, e. g., by a general rule-based system.

The system focuses on collaborative problem solving and is based on the *SharedPlans* theory of discourse (Grosz and Sidner, 1986, 1990). The interaction is highly constricted: the user can speak or make a menu selection from a set of possible utterances possible at a point in the interaction (see figure 3.2; the interaction menu is the window in the lower left corner). The user's interactions with the application are observed by the agent. In parallel, the agent can interact with the application with a cursor of its own. While a shared plan is pursued, the system agent and the user may perform interleaved steps; also, several plans may be followed simultaneously. The task model contains a set of "recipes" consisting of a sequence of partially ordered steps towards a goal, and pattern-action rules to trigger the recipes. The test application features 8 recipes for 15 different goal or action types. Some recipes in



How should a trip on the route be scheduled?

Propose scheduling a trip on the route via ___.
Stop proposing how a trip on the route be scheduled.
Retry proposing how a trip on the route be scheduled.
Undo proposing how a trip on the route be scheduled.

Figure 3.2: *COLLAGEN* test application screen (Rich and Sidner, 1998)

COLLAGEN are also specified non-declaratively, i. e., hard-coded methods generate a recipe for a given objective procedurally.

COLLAGEN implements no general planning or plan recognition approach for discourse interpretation. Instead, only the steps of a current recipe and recipes known for the current segment's purpose are examined non-recursively. To encode the communication acts realized by the user and the agent, *COLLAGEN* uses an artificial discourse language (Sidner, 1994), of which only two act types are included in the system described in (Rich and Sidner, 1998), *making* and *accepting proposals*. For discourse interpretation, an algorithm by Lochbaum was implemented (Lochbaum, 1998). First, the system tries to classify acts to be in one of five categories. An act either (1) directly achieves the current purpose, (2) is one of the steps in a recipe for the current purpose, (3) identifies a recipe for the current purpose, (4) identifies who should perform the current purpose or a step in a recipe for it, or (5) identifies an unspecified parameter of the current purpose or a step in the current recipe. Otherwise, the act is considered an interruption. The algorithm then adds the act to the current discourse segment on top of a focus stack, and if the act completes the dialogue segment purpose, the segment is popped off the stack. Discourse generation reverses Lochbaum's algorithm.

The system records the decisions of the agent and, based on context, provides an agenda of expected communication and manipulation acts based on the aforementioned five cases. This agenda is available to the agent and (in part) to the user. For communication, a selection from utterances that are generated corresponding to expectations of the system is used, as a replacement for natural language understanding. (this also means that it is not possible for the user to input utterances that are not expected). A history of the communication is generated and organized in a hierarchical tree structure of discourse segments and subsegments, which can be viewed by the user. The history display also includes the unexecuted steps of recipes, and thus provides information about the expectations of the system.

The user or the system agent may seize the initiative at any point. There is no explicit notion in *COLLAGEN* of obligations in discourse, instead it is up to the agent to decide when and how it is relevant or appropriate to contribute at any point in the discourse. Likewise, there are no mechanisms provided for turn taking, control or grounding (although Rich and Sidner experimented with some ad hoc mechanisms, such as waving the "hand" of the agent to gain attention of the user). Using the discourse history, it is possible to apply transformations that allow, e. g., to rewind the conversation to an earlier point, to replay part of the conversation, or to explicitly manipulate the focus stack for other purposes.

Unlike our system, *COLLAGEN* is not primarily concerned with dialogue management and leaves aside the internal reasoning of the communicating system agent. The collaboration manager takes a similar role to the CDE controller used in our framework described in Chapter 6, in that it comprises an outside mediator between the communicating parties, and their interaction with a "task world" separate from the participants (i. e., the application). Another similarity is that the dialogue history is used to generate expectations about possible future acts of the participants.

3.2.3 *RavenClaw*

The *RavenClaw* dialogue management framework that was developed as a successor of the *AGENDA* architecture used in the *CMU Communicator* (Bohus and Rudnicky, 2003). It is meant to provide a basis for constructing task-oriented dialogue systems using a set of predefined, domain-independent conversational behaviors. The idea is to reduce the system designer's concerns to the specification and maintainance of the actual task. *RavenClaw* then uses the task specification to automatically generate dialogue strategies. Consequently, the system places special emphasis on a clear separation of discourse and task knowledge.

To build a full natural-language interface, *RavenClaw* requires additional modules for, e. g., speech recognition and natural language understanding; it has mainly (but not exclusively) been used with *Olympus* dialogue system architecture that employs a classic pipeline processing order and is also a descendant of the *CMU Communicator* project. A number of systems for quite diverse scenarios have been successfully realized with *RavenClaw*/Olympus.[2]

RavenClaw manages a multi-agent system that uses a combination of *fundamental agents* and *agencies*. In the tree shown in figure 3.3 (adapted from (Bohus and Rudnicky, 2005)), fundamental agents correspond to leaf nodes, agencies to non-terminal nodes. The former are capable of realizing one of four basic operations: *Inform* is used to present information, *Request* to ask for information, and *Expect* accepts input without requesting it. The fourth kind is called *DomainOperation* for other types of domain-related operations. Agencies are agents that control other agents on a higher level, e. g., the *Login* node. The agents comprise preconditions, execution routines, and completion criteria. The dialogue engine operates in interleaving execution and input phases and allows mixed-initiative interactions (some of the

[2]An overview of these—along with a downloadable version of the framework itself—can be found on the website of the project, *http://www.cs.cmu.edu/∼dbohus/ravenclaw-olympus/systems_overview.html*

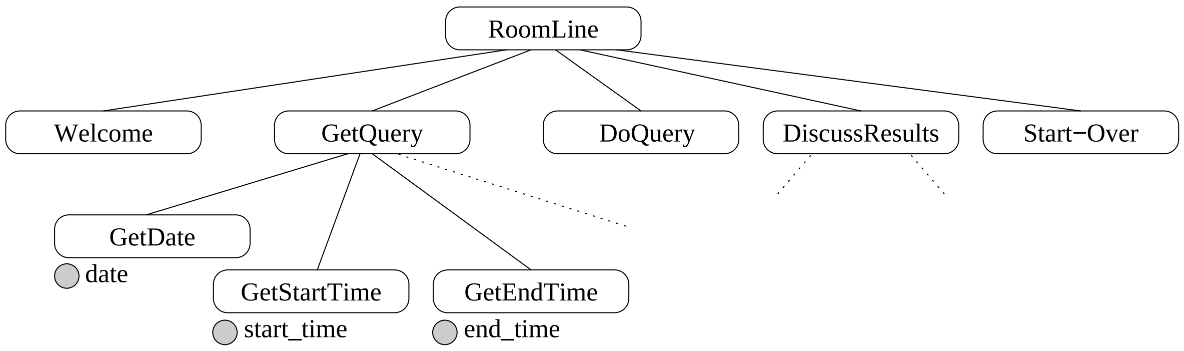


Figure 3.3: *RavenClaw*: Hierarchy of agencies and agents

implemented systems are, however, system-initiative only). The dialogue engine also uses expectations to disambiguate input.

The *RavenClaw* framework also focuses on ensuring robustness and error handling strategies that are designed to be reusable for different applications. An independent error handling process runs parallel to the application agents and can be configured to apply different handling strategies. In case of misunderstandings, for example, some of the possibilities are to ask the user to repeat her utterance, or to rephrase it, to re-state the prompt verbatim or in more verbose form, to notify the user of the misunderstanding, or to do nothing (Bohus and Rudnicky, 2005).

Like our framework, the *RavenClaw* system aims to be a general-purpose environment for dialogue systems. It also features a library of multiple agent types for different purposes that can run concurrently in a process hierarchy (but it does not realize multi-party interaction. As exemplified by the agents that implement different error handling strategies, it supports parameterizing an application by using different application-independent building blocks.

3.2.4 *WITAS* and other *TrindiKit*-Related Systems

TrindiKit is a collection of tools for experimenting with dialogue move engines and information states (Traum and Larsson, 2003; Larsson and Traum, 2000). The toolset has been and continues to be used and developed further in the course of several projects. These include the original *TRINDI* project (for "Task-Oriented Instructional Dialogue"), *DIPPER* (Bos et al., 2003), *GoDIS* (Larsson et al., 2000) and *IBiS* (Larsson, 2002), *SIRIdUS* (Kruijff-Korbayová et al., 2003), *WITAS* (Lemon et al., 2002) and the *TALK* project (Ljunglöf et al., 2005; Kruijff-Korbayová et al., 2006). The *TrindiKit* is based on the principle of *information state update* using update rules (cf. section 2.4.3.3). It aims to be a general framework for testing and developing dialogue theories rather than presenting a single theory (Kruijff-Korbayová et al., 2006). As an example, we take a closer look at the *WITAS* system.

WITAS The *WITAS* system is used to remotely control an autonomous small robot helicopter UAV (see figure 3.4) in real time.[3] The helicopter carries deliberative and perceptive systems on-board. During flight, new objects appear and it is possible for the operator to refer to them in the dialogue and, e. g., direct the helicopter to fly towards an object, or to follow an object. *WITAS* consists of a set of modules communicating by an open agent architecture (OAA2) facilitator. The system features spoken input and TTS output; a GUI display showing the current environment allows to use deictic references. The system uses the *Nuance* speech recognizer, *Festival* TTS, and the *Gemini* parser and generator. The generative component is designed to be symmetrical to interpretation, i. e., to only generate utterances that it could also parse. This way, the user is primed to speak "in-grammar".

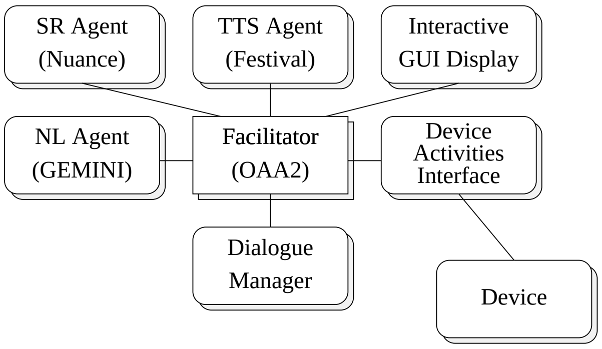


Figure 3.4: *WITAS* helicopter and system architecture

Several concurrent activities are possible. The interleaving of multiple communication threads about activities of the robot, to discuss several issues simultaneously, is a research topic of *WITAS*. The goal is to provide an interface that allows control of the vehicle by a non-expert using standard English, without requiring a specialized command-and-control language. At the same time, the system also has to be be sufficiently robust, since errors can easily result in damage to the moving robot. A related requirement is transparency in how spoken commands have been understood by the system; therefore, the helicopter always gives explicit feedback appraising its understanding.

The declarative descriptions of the goal decomposition of activities use *activity models*; the current and planned activities are accessible via an *activity tree* similar to a Hierarchical Task Network that contains a representation of the current and scheduled activities. The atomic actions the system can execute are dialogical and also physical actions. They are represented in a logical form (e. g. *(locate, np[det(a), truck])* for the utterance *"locate a truck"*). The dialogue acts are described by a set of abstract dialogue move classes such as *command*, *wh-question*, etc. The logical forms of the input are tagged with such a dialogue move. Incoming utterances are interpreted relative to a current *dialogue move tree* (DMT) which specifies which utterances can be interpreted in the current context and how they are to be interpreted. (Lemon et al., 2002) describe the operation upon the dialogue tree as a variant of "conversational games". Besides the DMT/activity tree approach, there is also an effort to use case-based reasoning to control the actions of the robot (Eliasson, 2005).

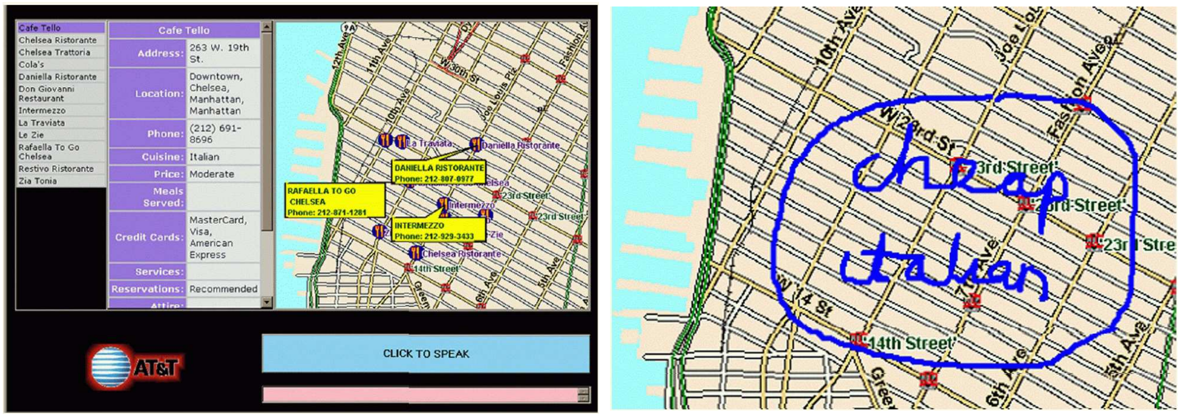
[3]The images were taken from the project websites *http://www-csli.stanford.edu/semlab-hold/witas/* and *http://www.cvl.isy.liu.se/Research/Robot/WITAS/blobs.html*

TrindiKit is an influential model for the storage and manipulation of the dialogue information state, a task that is required for any non-trivial dialogue system. There are many differences between the systems that have been realized with the toolkit (which emphasizes its versatility). It does not prescribe much about the way reasoning should be done beyond the use of update rules and an update strategy; approaches using production rules, dialogue move trees, or case-based reasoning have all been implemented.

For the use cases this thesis is concerned with, the *WITAS* system is relevant because it is also dependent on time-critical action, which includes dialogical as well as physical action. The requirement to honor real-time constraints is very strict, failure to do so would endanger the helicopter's safety on a mission. The system also has to process and interpret real-world percepts, and therefore has a multimodal component (although not with respect to user interaction).

3.2.5 *MATCH*

MATCH ("Multimodal Access to City Help") (Johnston et al., 2001) is a multimodal system that runs on a portable device (a *Fujitsu* PDA). It offers information about restaurants and subway stations in New York City (see figure 3.5). The user can interact via a combined voice and pen interface and the system answers with text-to-speech and a browser-based graphical output. The restaurant information part can provide information about restaurants in a particular area and their attributes (cuisine, pricing, etc.). The subway route application can compute an itinerary from a given map point to another and generate a dynamic presentation for it (there is no incremental guidance functionality, though).



Cafe Tello
Chelsea Ristorante
Chelsea Trattoria
Cola's
Daniella Ristorante
Don Giovanni Restaurant
Intermezzo
La Traviata
Le Zie
Rafaella To Go Chelsea
Restivo Ristorante
Zia Tonia

Cafe Tello	
Address:	263 W. 19th St.
Location:	Downtown, Chelsea, Manhattan, Manhattan
Phone:	(212) 691-8696
Cuisine:	Italian
Price:	Moderate
Meals Served:	
Credit Cards:	MasterCard, Visa, American Express
Services:	
Reservations:	Recommended

Figure 3.5: *MATCH*: left: user interface, right: unimodal selection on the map

The system supports different types of gestures (circling, line drawing, pointing, etc.). The user can talk about more than one entity by, e. g., circling several restaurants and asking about their phone numbers, or putting a constraint over an area of the map, as shown in the right image. The user has to indicate the start of an utterance boundaries by a click with the pen. This is justified by decreased susceptibility to background noise in outdoor use, but also facilitates segmentation of the input in turns. The dialogue model of *MATCH* is agent-based with a finite-state automaton model that is compiled to cover all possible modality combinations.

MATCH features the integration of multiple modalities, but does factor out issues related to asynchronicity by requiring the user to delimit turns. The task domain is relatively small; for more complex tasks or interaction sequences, the finite state representation would probably not scale well and be difficult to understand and maintain because of the size of the compiled automaton.

3.2.6 *QuickSet*

QuickSet (Oviatt and Cohen, 2000; Cohen et al., 1997) is a system that can run on a different platforms, including handheld devices, desktop systems, or fully-fledged augmented reality environments (cf. Figure 3.6).[4] It provides multimodal interaction using speech and gestures using a pen device. One research goal is to investigate under what circumstances different modalities are most useful and/or actually used. The system has been used with different backend systems, chiefly map-based military simulations (*LeatherNet* and *ExInit*); there is also a civil information seeking application called *Mimi* in which the users can query information about medical centers on a city map. Several users can interact together in a scenario with shared views (Cohen et al., 1997). The primary component of the user interface is a geo-referenced map, which can be panned and zoomed, and the users can annotate the map by marking points and lines. The users place military units and control measures (e. g., objectives) on positions on the map for simulation purposes.

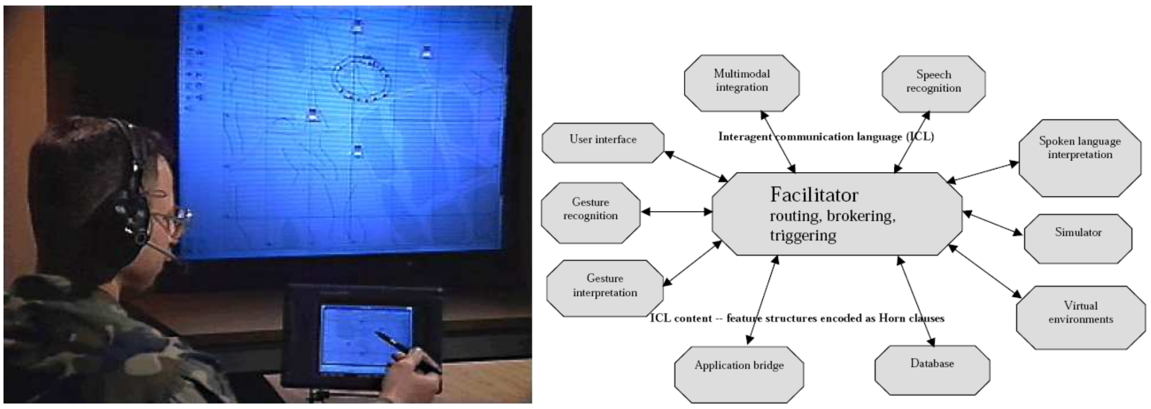


Figure 3.6: *QuickSet* simulation setup (Cohen et al., 1996), illustration of the facilitator (Oviatt and Cohen, 2000)

The system uses the OAA communication architecture, with a blackboard and registration / subscription mechanism. The recognized input is encoded in messages of typed feature structures by a late, semantic-level fusion component using unification (Oviatt, 2002). (Cohen et al., 2000) analyze the performance of test subjects with this setup and find a speedup for interactions using this multimodal interface compared with a standard direct-manipulation GUI, as well as a strong preference for using multimodal interaction among the test subjects.

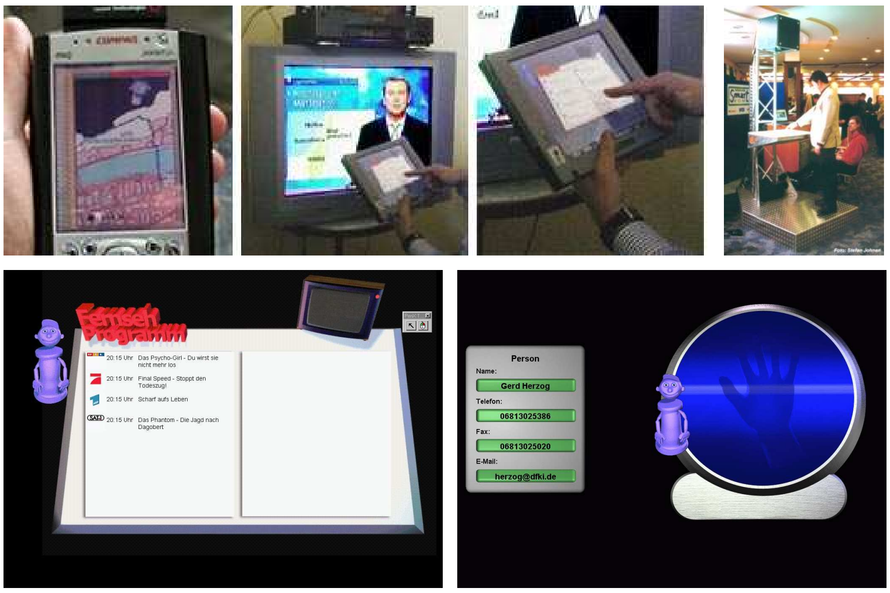
[4]In the mobile-use case, the bulk of the processing is shifted to a remote system for performance reasons.

The interaction follows a *command-and-control* paradigm: the user always has the initiative in the interaction, and user input essentially consists of commands that the system presents the results of.

An important result of *QuickSet* is the extensive analysis of system runs with regard to preferences, efficiency, and ratio of multimodal interactions (Oviatt, 2000; Oviatt and Cohen, 2000). Among the results are that a large advantage of multimodality lies in superior error handling, both in terms of error avoidance and graceful recovery, e. g., due to the possibility of mutual disambiguation (Oviatt, 2002). It was also found that in the context of the collected data, the presumption that there has to be temporal overlap between components of a multimodal construct was not justified, and that complementarity of content from different modalities was more common than redundancy.

3.2.7 *SmartKom*

SmartKom (Wahlster, 2003a, 2006; Reithinger et al., 2003) is a multimodal, task-oriented dialogue system. It features multiple applications, 13 in total, of which most are using and relying on services of other applications. The user interacts directly, in mixed-initiative, with a personification of the system, called Smartakus, to whom she may express her wishes. Smartakus will then look for a way to accomplish the task that has been delegated to it. This setup is called the *SDDP metaphor* (for **S**ituated **D**elegation-oriented **D**ialogue **P**aradigm, see (Wahlster et al., 2001)).



20:15 Uhr Das Psycho-Girl - Du wirst sie nicht mehr los
20:15 Uhr Final Speed - Stoppt den Todeszug!
20:15 Uhr Scharf aufs Leben
20:15 Uhr Das Phantom - Die Jagd nach Dagobert

Person
Name: Gerd Herzog
Telefon: 06813025386
Fax: 06813025020
E-Mail: herzog@dfki.de

Figure 3.7: The *SmartKom* system: (Top) the different platforms *mobile*, *home* and *public*, (bottom) screenshots of the EPG and biometrics applications

The system's avatar, Smartakus, is rendered as a cartoon character rather than a life-like one. Nevertheless, he features a repertoire of human-like expressions to convey its current state

(e. g. he will cap his ears in some situations to make clear that he expects the user to say something, and has animations to show that he is currently working on something).

The system is deployed on three different hardware platforms (see figure 3.7): a stationary kiosk, a mobile handheld device, and a portable tablet PC for use in the home. The application configuration is changing with the hardware platform. Reasons for this are hardware requirements (e. g., a scanner necessary for fax service and biometric hand recognition is only available in the kiosk hardware setup) as well as applicability (the incremental route planning application only makes sense with a mobile device).

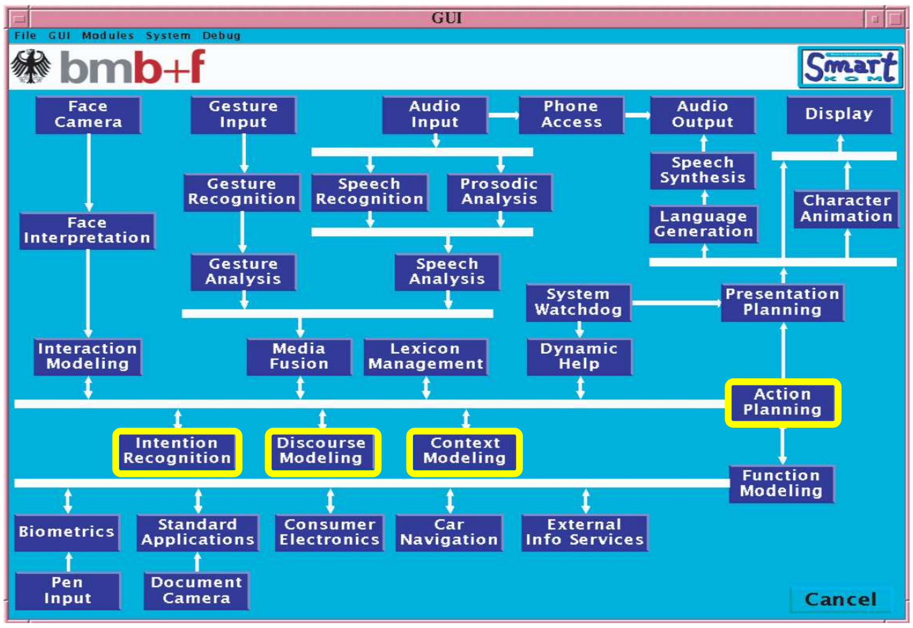


Figure 3.8: The *SmartKom* architecture (dialogue management modules are shown with highlighted borders)

SmartKom is heavily modularized and uses a hub-and-spoke architecture for communication between the modules (figure 3.8). The communication hub, *Multiplatform* (Multiple Language / Target Integration Platform for Modules), makes it possible to integrate modules running on different hosts, operating systems, and language environments to be interconnected. It was specially developed for the system (Herzog et al., 2003).[5] The roles of the modules allow them to be put into functional groups respective to the responsibilities of input/output, dialogue processing, and application modeling.

The domain and knowledge of *SmartKom* is modeled in an ontology that is transferred to an XSD/XML representation in the Multimodal Markup Language (M3L) by specially tailored tools, as described in (Porzel et al., 2003). The use of a rigorously structured ontological representation facilitated the development of the large knowledge base of about 730 concepts and 200 inter-concept relations enormously, and provided for a smooth interfacing of the many components of the system, along with offering sufficient expressive power.

[5]*Multiplatform* is available at *http://multiplatform.sourceforge.net/* as open source software.

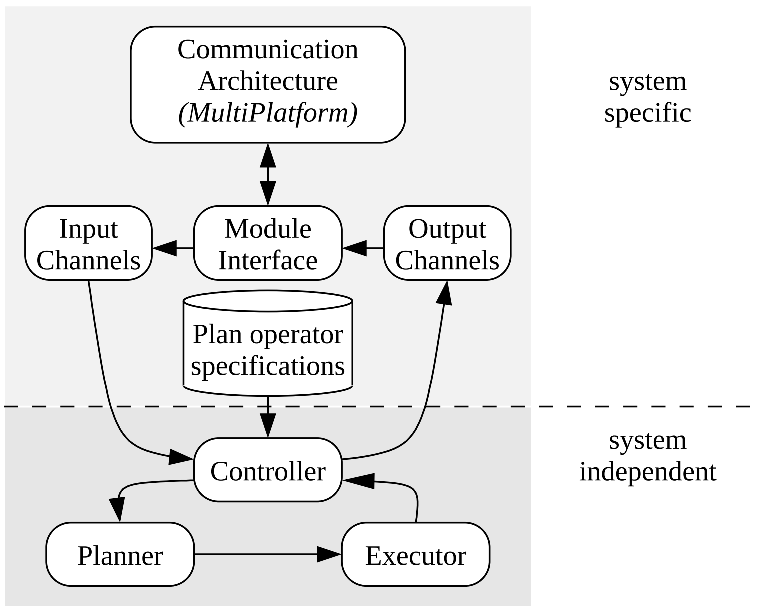


Figure 3.9: Internal structure of the the *SmartKom* action planner

In *SmartKom*, dialogue management is distributed between four modules addressing the subtasks of intention recognition, discourse modeling, context modeling and action planning. The action planner uses task specifications called "action plans" that contain a collection of system-specific goals and plan operators for the different applications, and comprises a controller that operates a forward planning and execution monitoring algorithm and communicates with the rest of the system via abstract channels (Figure 3.9). Expectations about future moves of the user are delivered to the multimodal analysis modules and the discourse modeler. While the general concept of expectations was already introduced in earlier systems (e. g., *RavenClaw*), the expectations in *SmartKom* are novel in that they are more fine-grained and address several aspects of input. (Löckelt et al., 2002) outline the protocol and the informational structure of the expectation mechanism. The expectations in *SmartKom* inform the analysis components about which slots of active domain instances are expected to be referred, as well as about the currently active goal of the action planner. However, there is no account of the form in which they are expected to be addressed, e. g., by an explicit answer to a question or a statement by user initiative. Lexicon updates, depending on the context of the current task, are also incorporated into the expectations. One instance is the EPG application which continuously updates the dynamic lexicon with lexical items concerning movie titles and actors as they are retrieved in response to the user's browsing of the database, and made salient by displaying them on the screen.

The dialogue manager realizes flexible mixed-initiative dialogue (Löckelt, 2004, 2006). The system will take the initiative to collect additional information needed for carrying out tasks, as well as accept and integrate user-initiated dialogue moves. In one application, the interactive tour guide, system initiative can also be triggered in response to asynchronous external events. If a tour is in progress, the position of the user is constantly monitored via a GPS device. Certain landmarks in the location database are annotated with additional information,

e. g., if the place is of historical interest. When the user approaches such landmarks while the tour guide is active, the application initiates a corresponding event. The dialogue manager will then interrupt and suspend the currently active subdialogue and display background information about the location.

The basic structure of the conversational dialogue engines described in this thesis is based on the action planner component of *SmartKom*. Several techniques that were introduced in *SmartKom* are refined and extended in the multi-party scenario of *VirtualHuman*.

3.2.8 *SmartWeb*

The *SmartWeb* project provides multimodal question-answering interactions in open domains to allow a mobile and context-aware interface to the Semantic Web (Wahlster, 2004; Reithinger et al., 2005). It combines research efforts in multimodal interaction, Semantic Web technology, information extraction, and mobile devices for web access.

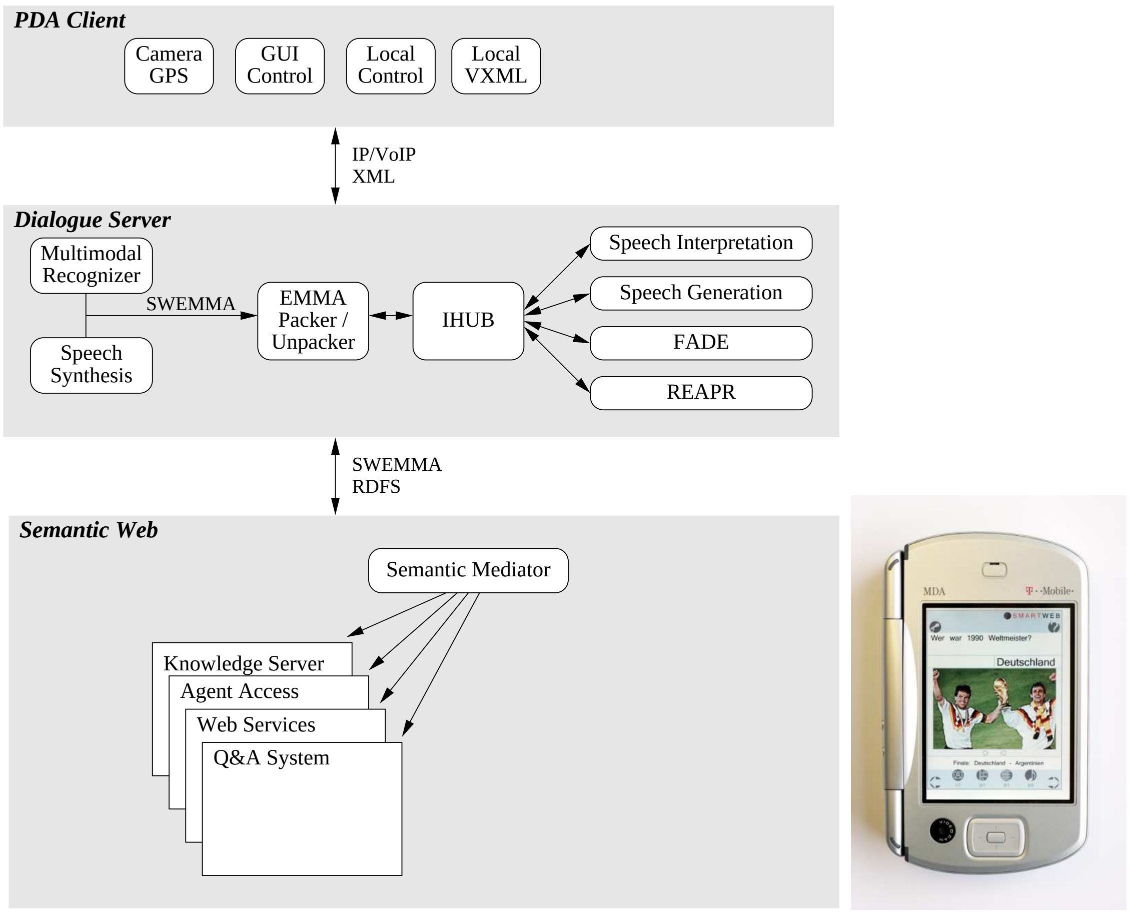


Figure 3.10: Architecture of *SmartWeb* handheld (Sonntag et al., 2007), *SmartWeb* on the MDA

A sketch of the architecture, using the example of *SmartWeb*-Handheld with a smartphone interface, is shown in figure 3.10. Other realized setups allow interaction in a car or while riding a motorbike. The modules of the system run in a distributed fashion, with more computation-intensive modules running remotely on a dialogue server. The dialogue server can host multiple instances of the system that are created on-demand when a remote device connects. In the dialogue server, the communication infrastructure is provided in a hub-and-spoke architecture called *IHUB* (Reithinger and Sonntag, 2005). The *IHUB* distributes messages between the dialogue modules for speech interpretation (*SpIn*), generation (*NipsGen*), the fusion and discourse engine (*FADE*) and the system reaction and presentation component (*REAPR*). The EMMA unpacker and packer (*EUP*) connects the *IHUB* to the multimodal recognizer, the speech synthesizer and the Semantic Mediator. The applications behind the Semantic Mediator can be addressed by a SOAP/WDSL API (Reithinger et al., 2005). *SmartWeb* features comprehensive ontological modeling using the W3C recommendation EMMA and RDFS. EMMA is used to encode multimodal I/O, and RDFS for interactions with the Semantic Web interface. The goal was to let different domain ontologies interoperate in a common data model; for this, the general-purpose ontologies SUMO and DOLCE were merged (Sonntag and Romanelli, 2006).

The initial use case scenario was that a user visits a World Cup stadium during the soccer championship in 2006, and has related queries regarding, e. g., soccer results, but it is possible to request information from any domain. The interaction comprises asking information-seeking questions, requesting additional services available by integrating external applications, and system control queries (e. g., canceling a running query or asking for status information). The user can also use the pen to edit recognition results, which is immediately presented on the screen after recognition, in case they are incorrect. The user can also correct recognition results by speech, e. g., *"I did not mean Fuji, but Fiji"* (Reithinger and Sonntag, 2005). On the output side, the system attempts to provide answers in several modalities, e. g., a typical reply to the user's question *"who won the soccer championship in 1990?"* would be a TTS output *"Germany"* accompanied by a picture of the winning team, as shown on the right side of Figure 3.10. The interaction follows a cycle of turns where *REAPR* first takes input from *SpIn* and *FADE*, and presents it again to the user for reconsideration. If the user makes a correction, the input is reprocessed by the interpretation modules. Otherwise, the request is sent to the Semantic Mediator. After the results arrive from the Semantic Mediator, they are presented on the device (Reithinger et al., 2005). Additionally, the user can interrupt the processing cycle by pressing a *Cancel* button.

For the *OMDIP* project, which is documented in Chapter 7, our dialogue manager was integrated with the IHUB communication infrastructure and used *SmartWeb*'s ontology framework. There was also cross-fertilization due to the parallel development of *SmartWeb* and *VirtualHuman*, both of which use the fusion and discourse engine *FADE* and a similar toolset for the development and maintenance of the ontological knowledge base.

3.3 Interactive Narratives

Systems that use dialogues to perform storytelling include many commercial applications, especially in the games industry. Since the mid-1970s, there have been games of interactive fiction. The early *Infocom* series already incorporated a sophisticated parser capable of processing simple anaphora and relative clauses in typed input.[6] However, even today the intended target hardware for most interactive games—computer systems in the private home—is operating with keyboard and mouse input modes only and usually does not offer support for spoken language processing, let alone more sophisticated means for multimodal interaction.[7]

In this section, we first look at a "game" for adults that is also intended as a research platform: *Façade* achieves depth of immersion in a convincing scenario. Our second example is the commercial game *The Sims*. Although the interactions in this game do not include spoken dialogue in the classic sense, the character modeling is interesting for virtual storytelling systems. The *Mission Rehearsal Exercise* is a research project that realizes an ambitious multi-character scenario that also includes narrative elements. We then take a look at the related *IN-TALE* system, which makes use of a separate director module to control the goals of the characters. We have to leave out several interesting systems. Some have been using existing commercial game architectures to provide the environment for experimentation, e. g., the *Unreal Tournament* engine in the *Mimesis* system (Young, 2001) and "Haunt 2" (Magerko et al., 2004), which also examines the use of virtual directors. The *FearNot* research system features an unusual scenario involving educating children how to cope with bullying situations (Paiva et al., 2004).

3.3.1 *Façade*

Developed by Mateas and Stern, *Façade* (Mateas and Stern, 2003; Mateas, 2002) is a recent milestone in interactional narrative systems. The user can interact with two virtual characters, playing the roles of a married couple, in their apartment. The user impersonates a visitor and is involved in a developing marital conflict situation. Although the interaction possibilities are relatively limited and the story quite constrained by the storytelling requirements, the system manages to convey a strong sense of immersion in the situation.

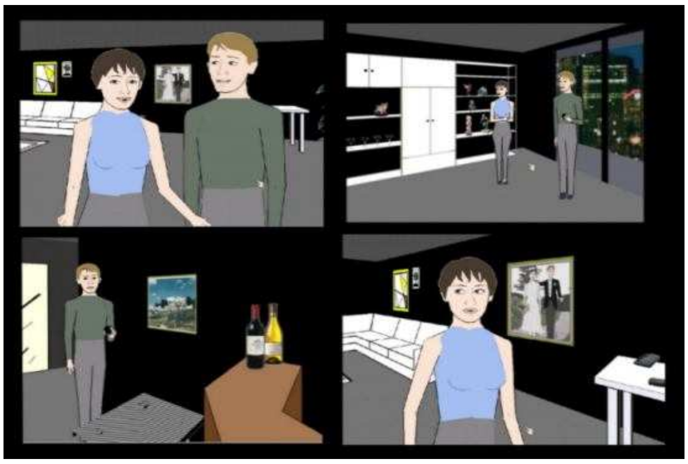


Figure 3.11: Screenshots from *Façade*

[6]Wikipedia article on *Infocom*: *http://en.wikipedia.org/wiki/Infocom*
[7]This is currently changing somewhat with the introduction of the *Nintendo Wii* gaming console.

The user can move around in the apartment of the characters which is rendered in comic-strip-like pseudo-3D, as displayed in figure 3.11. *Façade* recognizes a set of keyword inputs and mouse interactions with objects that trigger predefined story events. For example, if the user expresses interest in a painting on the wall by referring to it or by standing next to it for a longer period of time, the characters will start to argue about whether the location of the painting is suitable, and try to persuade the user to take their respective side in the argument. Failing to react to some of the characters' actions can also have consequences.

The system only processes typed keyboard input which is processed in real time. This introduces a disadvantage for the user into the dialogue situation, since her dialogue contributions tend to be lacking in speed or accuracy; the system does not compensate for slow typing or typing errors. Also, since the input interpretation is rather simplistically based on relatively few keywords, often an utterance is understood incorrectly. Both aspects can be a source of frustration, because the characters are quick to throw out the visitor in case they are not satisfied with her contributions, or if they—possibly by mistake—interpret a user action as inappropriate.

It is not really possible to engage in a more complex, meaningful conversation with the characters. The story unfolds in small scripted sequences tied to key events and the progression of time. However, the wording of the utterances and their triggers are cleverly chosen, so that the user gets the feeling that she is actually in a scene with an (admittedly egocentric and agitated) couple. Much of the atmosphere is created by the exchanges between the characters, not with the user, who acts mainly as the trigger for events. The lack of real control over the interaction, and resulting unintended consequences such as being thrown out of the apartment, is implicitly made more plausible by the context of a marital conflict where the user is in the role of a bystander anyway, and all participants are emotionally upset and may "over-react".

In *Façade*, a single user can experience real-time multi-party interaction with autonomous characters. However, it is more an example for storytelling systems than for dialogue systems. The system manages to enforce a storyline, with different yet defined endings, while allowing the user to act freely and get a sense of immersion.

3.3.2 *The Sims*

There have been two instalments of the series *The Sims*. *The Sims 1* was the best-selling computer game of all time with over 6.3 million copies sold worldwide, according to a statement by the publisher company *Electronic Arts* in 2002. The game is a psychological simulator set in a neighborhood of semi-autonomous characters whose interactions create an emergent narrative. There has been work to incorporate multimodal input via a tabletop interface (Tse et al., 2006), although in this case, the interaction did not include interactions with the characters themselves, but the overall environment of the game.

Although there is a sort of "dialogue" between the characters, along with synthesized "speech" output in a simple artificial language called "Simlish", the interactions are not intelligible to the human user. To hint at meanings, illustrative speech balloons with icons are displayed above the heads of the characters, as seen in Figure 3.12. The characters also use the additional modalities of gestures, body posture and facial expression to convey their condition and intentions.

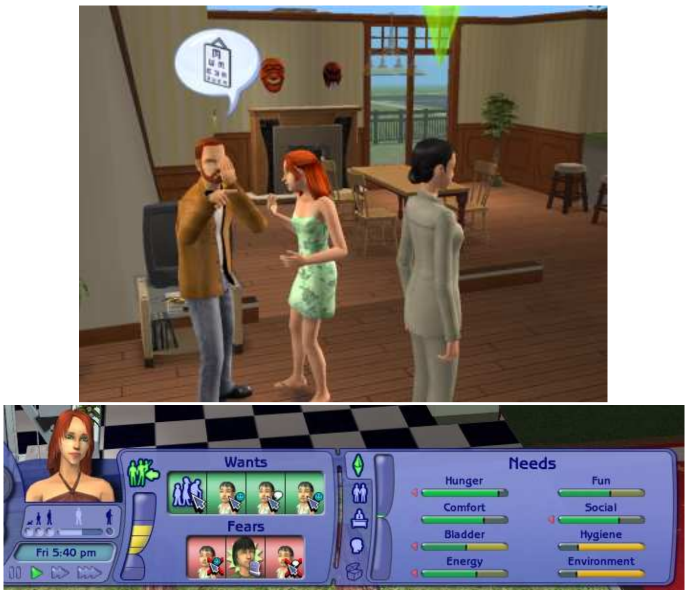


Figure 3.12: Screenshots from *The Sims 2*: (top) character interaction, (below) status bar displaying a character's wants, fears, and needs

Unfortunately (but not surprisingly), the creators of this commercial software are not very open about the inner workings of their product. However, from the interface it can be concluded that the characters in *The Sims 2* are modeled in a way that is in some ways remarkably close to concepts in the research literature about life-like communicating agents. For one, a Sim character is defined in terms of character traits that are similar to the psychological "big five" that also were used by the *ALMA* affect engine in *VirtualHuman* (see section 7.2.5). They include, e. g., degrees of "extraversion" and "neatness". The behavioral model for the characters, on the other hand, is reminiscent of BDI theory (see section 2.4.3). The characters have positive and negative desires (called "wants" and "fears" in the game) that are derived from the character traits (e. g., a "neat" character will eventually have a desire to wash the dishes).

The desires however do not directly lead to actions of the characters. What the character intends to do in a given situation can be determined by direct commands from the player, or arise depending on the desires or one of a set of "needs". Needs are quantitative measures of conditions such as "comfort", "hunger", or "hygiene" (see the lower part of figure 3.12). For example, if the "hunger" value sinks below a certain threshold, the character will adopt an intention to eat a meal. Intentions are added to a queue, and only one intention can be active at any one time. The currently active intention leads to actions to fulfil the intention, e. g., planning how to go to the kitchen, open the fridge, prepare a meal, and carry it to a suitable table to eat it. How exactly the intention execution is scripted or planned is not entirely clear, but at least the path-finding toward objects needed for the actions is flexible enough to dynamically accommodate moved objects or other characters standing in the way.

The way in which the user can interact with objects and characters is modeled alluding to the concept of *affordances* (see section 4.4.5). Possible actions involving an object are triggered by clicking on the object and choosing from a set of options, this set is provided by the object itself. As suggested in the section about affordances, this facilitates authoring new content in the scenario: New objects have been added by a very active player's community that come with customized actions available with them.

3.3.3 *Mission Rehearsal Exercise* and *IN-TALE*

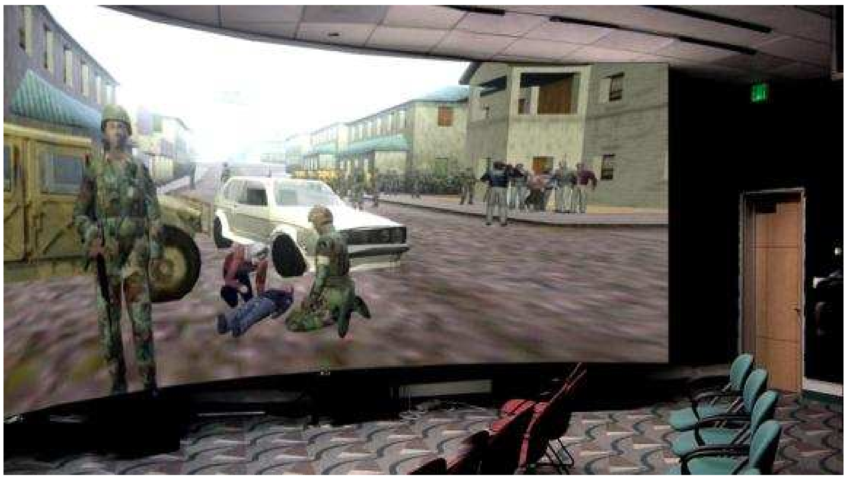


Figure 3.13: The *Mission Rehearsal Exercise* System

MRE The *Mission Rehearsal Exercise* (*MRE*) (Swartout et al., 2005, 2006) is an ambitious project carried out at the Institute of Creative Technologies (ICT). It simulates an emergency situation during a peacekeeping mission in Bosnia where the human user, a trainee, takes on the role of a lieutnant in charge of a team and has to coordinate squads of virtual soldiers to resolve the situation.

The scene is presented in 3D on a large screen (see figure 3.13) where the trainee can perceive the virtual environment and interact with life-like and life-size virtual characters taking the roles of different team members via speech. The characters were created by Boston Dynamics Incorporated (BDI) and include facial expressions and lip synchronization. The animation of the characters was done using a combination of motion-capture and procedural animation techniques. *MRE* uses a blackboard communication infrastructure called *Elvin* similar to the *IHUB* and *MultiPlatform*. *Elvin* is bypassed for time-critical tasks such as the synchronization of animation and audio to mitigate latencies, as well as for text-to-speech and gesture generation to achieve lip synchronization (Swartout et al., 2005).

The stated research goal of *MRE* is to improve capabilities such as perception, planning, spoken dialogue and the display of emotions, and to integrate it into a single agent architecture. The setting especially allows for interesting research topics related to multi-party dialogues. The interactions with the characters are influenced by the social roles in a military situation, which include the superior / subordinate relationship, professional roles (e. g., medic), authority to order a task and responsibility for a task to be carried out. Traum's investigation of issues in multi-party dialogues (Traum, 2004) is strongly guided by the *MRE* setup. Characters can move around in the virtual world, answer questions about the state of the world, suggest appropriate future actions, respond to orders by acting or counterproposing alternatives, engage in clarification sub-dialogues, and issue orders to subordinates. They also exhibit attending behavior (such as gazing) and can engage in multiple conversations (Traum and Rickel, 2002).

The behavior of the (up to three) major characters is autonomous, minor characters and physical events (e. g. explosions) are scripted. Scripted events can be triggered either autonomously (production rules) or by a human exercise controller. Characters have a world model that also includes assumptions about the intentions of other characters, and plans that specify who might be able to carry out sub-tasks. They feature behavior to express attention to other characters and objects, such as gazing. The autonomous characters are implemented using a production system in the *SOAR* architecture (Laird and Rosenbloom, 1996) and are partially based on *TrindiKit*.

In terms of setup, the *MRE* has requirements that are similar to those of the *VirtualHuman* system, especially regarding the number of virtual characters and the objective to let them enact realistic visible behavior. Although the theoretical background of the has been extensively analyzed, the exact extent of the system's implementation is not entirely clear. In (Swartout et al., 2006), the authors state that *"an initial version of the MRE system described in this paper has been implemented"*, allowing the user to interact with three virtual humans, and that the system was ported afterwards to an additional domain that only features a single virtual character. A more recent publication describing the completion of the entire system could not be found.

IN-TALE Another approach to interactive storytelling is implemented in the prototype *IN-TALE* system, also developed at the ICT. Like with *MRE*, the scenario for this system is a training exercise for military personnel.

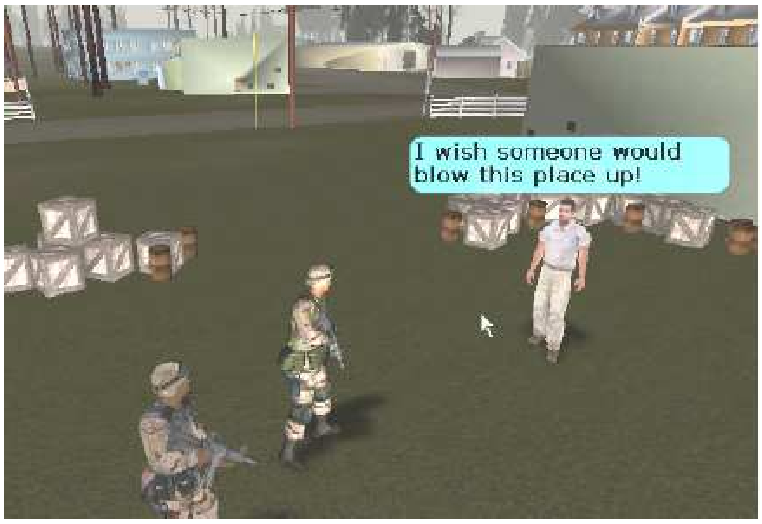


Figure 3.14: Screenshot from the *IN-TALE* prototype

The scenario is a peacekeeping mission in a foreign country where the trainee is put in a situation in which there is a threat of insurgents detonating a bomb in a marketplace (Figure 3.14 shows a screenshot). The situation involves characters that are hostile to each other, and it is unclear who sets up the bomb because there is more than one suspect. The trainee is not required to solve or prevent the criminal case; instead, the emphasis is to put him into a learning situation where he can practice relevant skills.

The characters' actions are scripted with the *ABL* language ("**A** **B**ehavior **L**anguage") that was also used in the *Façade* system (Riedl, 2005). However, *IN-TALE*'s focus is more on the narrative aspects of the system. The system employs a director module that is meant to ensure that the story goals unfold as planned. The intention is to realize believable actions by the characters and the ability to reconcile actions by the player that might threaten the desired end result, e. g., if the player arrests the insurgent before he has the chance to place the bomb, the outcome may be changed. One possibility in such a situation is to let narrative-threatening actions fail in a way that does not destroy the suspension of disbelief (Riedl and Stern, 2006b). In the bounds of the high-level goals from the director module (called "narrative directive behaviors"), the characters execute low-level autonomous behaviors (called "local autonomous behaviors"). In both cases, ABL scripts are triggered by matching associated preconditions against working memory elements (Riedl and Stern, 2006a). The cited publications do not go into the set of supported input/output modalities, which are in any case not the topic of the research. From the screenshot, it appears that textual output is generated on-screen, and as a game engine was used to render the scenario (Riedl, 2005), the input can be assumed to be genre-typical mouse interaction.

At the time of writing, *IN-TALE* is still a prototype system. The approach to let the high-level narrative goals be directed by an external module while letting the characters exert local autonomy is similar to *VirtualHuman*.

3.4 Other Systems and Approaches

For many simpler applications, using *finite state automatons* (FSAs) to represent the dialogue model, like in the *MATCH* system, is sufficient. In this case, the application is laid out as a finite set of states, or nodes, that are interconnected in a directed graph. The connecting edges represent transitions between states. Graph edges can be labeled with conditions or probabilities determining when a certain transition is allowed. On entering a node, one or more actions are executed. An early important system using scripted interaction is the *PPP* system realizing life-like presentation agents (André et al., 1998). *SceneMaker* is a toolkit for designing and executing FSAs for scripted characters, which was used to define multiple-character interactions in, e. g., the *CrossTalk* (Rist et al., 2002) and *COHIBIT* systems (Gebhard et al., 2003; Klesen et al., 2003).

Another alternative for less complex applications is to use *VoiceXML* interpreters for processing. *VoiceXML* (Voice Extensible Markup Language) is a standard introduced by the W3C consortium. It allows the dialogue designer to specify a voice-controlled application in a state-based manner. An interpreter that parses *VoiceXML* specifications can then render the application. The main uses of *VoiceXML* are for phone-controlled services, like form-based tech support, voice e-mail access, or package tracking. *VoiceXML* can also be used to direct a customer towards a specific human phone operator by asking decision queries about the nature of a call. A *VoiceXML* application consists of related documents that are interpreted by a “voice browser” analogous to HTML documents. *VoiceXML* presents a relatively easy way to specify dialogues that do not require advanced features. The language has found widespread use and, according to the specification webpage (W3C-VoiceXML, 2006), deployed *VoiceXML* applications in 2006 handled “millions of phone calls every day”. There is also a proposal by the W3C to extend *VoiceXML* for multimodal dialogue requirements.[8] (Niklfeld et al., 2001) argues that *VoiceXML* lacks adaptation capabilities to user and situational attributes for multimodal and multi-platform applications, which are important especially for mobile devices. The specification also is quite inflexible and not very powerful for more complex dialogues. VoiceXML dialogues for example do not allow the user to be proactive, i. e., to give information that is not expected at the current point of the interaction, but “scheduled to be addressed at a later time”.

Predominantly in non-research systems with a limited scope, or systems that are not primarily concerned with dialogue management per se, dialogue specification is also sometimes done using programming languages specially designed for a particular application. Storytelling systems often fall into this category. One example is ABL, which was mentioned in the sections about the *Façade* and *IN-TALE* systems. Special-purpose languages allow a dialogue designer to tailor and optimize to cater to the idiosyncratic demands of an application, but also have the drawback that they tend to trade-off theoretical soundness for pragmatic reasons; this can, e. g., entail that it is difficult to incorporate additional linguistic issues that were not originally planned for. Also, a re-use in different domains can turn out to be difficult for the same reason.

An avenue of research of growing importance with respect to task-oriented systems is the dynamic integration and easy or automatical configuration of multiple applications. Systems that comprise more than one application and possibly dynamically changing and/or interdependent application contexts raise additional integration and coordination problems that also affect the dialogue management modules. Examples for integrated multi-application systems are, e. g., the *EMBASSI* project which aims to provide an easy-to-use plan-based interface to networked information systems and home control (Heider and Kirste, 2002; Krämer and Bente, 2002). Another approach is the *DyMaLog* framework that allows *application-blind* dialogue processing with the *AIDE* dialogue engine and an object-oriented input representation (called o^2I-trees) of application parameters derived from an ontological knowledge representation. This representation allows in some cases to automatically infer cross-application dependencies (te Vrugt, 2006). *DIANEXML* is an XML-based specification language that uses a set of transactions, parameters, and a grammar. An application is identified with a set of possible transactions. *DIANEXML* was extended in (Dongyi, 2006) to a meta-description language that makes it possible automatically or semi-automatically analyze and combine the functions of existing speech user interfaces. Such possibilites for meta-specifications that span multiple applications could become increasingly useful to achieve interoperability between a mounting number of deployed, but mutually incompatible speech-enabled systems.

[8]see *http://www.w3.org/TR/multimodal-reqs*

3.5 Summary

This chapter has presented a selection of systems that concentrate on different aspects of human-computer conversational interaction in task-oriented and narrative-oriented domains, and use very different knowledge and dialogue management techniques, modality combinations, and theoretical backgrounds. Figure 3.15 juxtaposes some of the key features of these systems for comparison, and also lists the ones that our framework aims to provide.

Since beginning of *TRIPS* and the *SmartWeb* project, task-oriented systems became increasingly usable and practical to realize. At the same time, they are involving ever more complex tasks, multimodal I/O configurations, and domain descriptions. On the other hand, constructing a dialogue system still is a daunting undertaking. In the face of more comprehensive domain and task descriptions, the use of ontologies to organize them in a standardized way becomes more important, and so does the need for comprehensive tools for their creation and maintainance. The availability of multimodal interaction possibilities and free mixed initiative enhances the convenience, versatility, error-tolerance and efficiency of task-oriented systems. In addition, it makes the interaction feel more natural. Difficulties with the feasability of full logic-based approaches have cast light on the importance of using the "appropriate paradigm for the job", one that is powerful and expressive enough to handle the demands of the task, but also computationally tractable.

In the area of storytelling, there is not currently a strong emphasis on extending the dialogue capabilities of the characters. One reason for this may be that this feature is hard to realize in a completely or near realistic fashion, and unconvincing behavior on the part of the characters would endanger immersion and believability, and thus hurt the story. Efforts instead concentrate on making the characters act believably. This comes often at the price of much hard-coded, non-reusable work to cover possible situations with little real reasoning. In the extreme, problems with the combinatoric explosion of possible situations results in "interactive movies" with very few meaningful interaction possibilities, or incoherent and arbitrary "emergent" stories. There is also increasing research effort to provide authoring software for interactive narratives, e. g., the *Storytron* and *Erasmatron* systems (Crawford, 1999) and the authoring platform *INSCAPE* (Göbel et al., 2005). However, there is still a lack of tools that support creative writers on the non-technical level.

System	Multimodal Input / Output	Initiative	Dialogue Management Paradigm	Applications / Scenarios	Multi-Party Interaction	Knowledge Representation
TRAINS/ TRIPS	speech, typed / deep generation, GUI	mixed	planning	one at a time	no	logic
COLLAGEN	menu selection, speech, gesture / written and GUI output	mixed but arbitrary	SharedPlans recipes	(one)	no	logic
RavenClaw	(diverse)	mixed / system	agency tree	one at a time	no	agents and agencies
WITAS	spoken / TTS and GUI	mixed	theorem prover	one	no	*TrindiKit*
MATCH	speech and pen gestures / TTS and GUI	user	finite state automaton	two	no	finite state automaton
QuickSet	speech and pen gestures / GUI	user	command interaction	one at a time	two collaborating users	feature structures representing commands
SmartKom	speech, pen, facial recognition, pointing gestures / deep multimodal generation, gesture, and GUI	mixed	dialogue games, planning	multiple, interacting	no	ontology and plan operators
SmartWeb	speech and pen / TTS and GUI	user	Q/A cycle with corrections and modality selection	one	no	ontologies
Façade	typed keywords / recorded speech and GUI	mixed but arbitrary	event-triggered scripts	one	one user, two characters	production rules
MRE	speech / multimodal generation, character behavior	mixed	scripts and human controller	one	one user, multiple characters	*SOAR* / *TrindiKit*
IN-TALE	mouse (?) / written text (?), character behavior	mixed	scripts, external direction	one	one user, multiple characters	*ABL* special-purpose language
The Sims	mouse commands / artificial canned text, character behavior	(mixed)	unknown, at least partially planned	one, parameterizable	one user, multiple characters	(unknown)
CDE use cases	speech, pointing gestures / deep generation, TTS, GUI, character behavior	mixed	dialogue games with or without planning, Narrative Mode	one at a time	arbitrary many users and characters	ontologies, games, activity specifications

Figure 3.15: Feature comparison of related systems

Chapter 4

Representing the Knowledge Base for Situated Conversational Characters

4.1 Introduction

A conversation manager needs a way to represent the content of items of discourse, facts about the environment in which the conversation occurs, and the application or theme the dialogue is about. If it is concerned with a multi-party situation involving one or more virtual dialogue participants, it also needs to account for the mental state and reasoning capabilities they use to produce utterances and other behavior, as well as one or more user models. The decision about the representation method to use for holding and manipulating this information about the world has far-reaching consequences, since almost all parts of the dialogue system are in one way or another related to it. Ideally, the different modules of a system should share a common representation.

The system's knowledge base determines what is known to exist, what can be talked about, and what inferences can be drawn. Possible approaches include procedural representation, custom special-purpose representations, classical logic representations, frame-based representations, or ontology-based representations. The distinction between these representation methods is blurred, since they often are at least partly equivalent. For example, ontological relations can be transformed to equivalent logic expressions. However, there are still important differences, which we will explain below.

In this chapter, we first describe the types of knowledge that are needed in a dialogue system and the different purposes for which the knowledge is used. We look at ways to represent knowledge and justify our decision to employ an ontology-based approach, with a notation based on typed feature structures, and introduce important operations on ontological objects. The environment and the characters are each associated with a self-contained view of the world representing the context. We describe the notion of context in another section. Finally, a section talks about important concepts in the domain.

4.2 The Role of Knowledge in a Dialogue System

4.2.1 Levels of Knowledge Representation

(Liew, 2007) gives the following explanation of the distinction between data, information and knowledge (emphasis in the original):

> ***Data*** *are recorded [. . .] symbols and signal readings [. . .]* ***Information*** *is a message that contains relevant meaning [. . .] from both current (communication) and historical (processed data or 'reconstructed picture' sources) [. . .]* ***Knowledge*** *is the (1) cognition or recognition (know-what), (2) capacity to act (know-how), and (3) understanding (know-why) that resides or is contained within the mind or in the brain.*

These relations are sometimes also depicted in a pyramid that has data as its foundation, the next level constitutes information describing the data, and above is the knowledge that can be inferred from the information; another level can be added for *wisdom* derived from the knowledge, yielding a so-called "DIKW" hierarchy (Ackoff, 1989). In a multimodal dialogue system, interpretation and modality fusion modules are responsible for converting raw input data from the recognizers for the different modalities to messages representing, e. g., the meaning of user utterances (cf. Section 2.5.3 on dialogue system architecture). The conversation manager needs knowledge that defines its ability to interpret the information content of the messages representing communicative and other actions from other dialogue participants and to react accordingly.

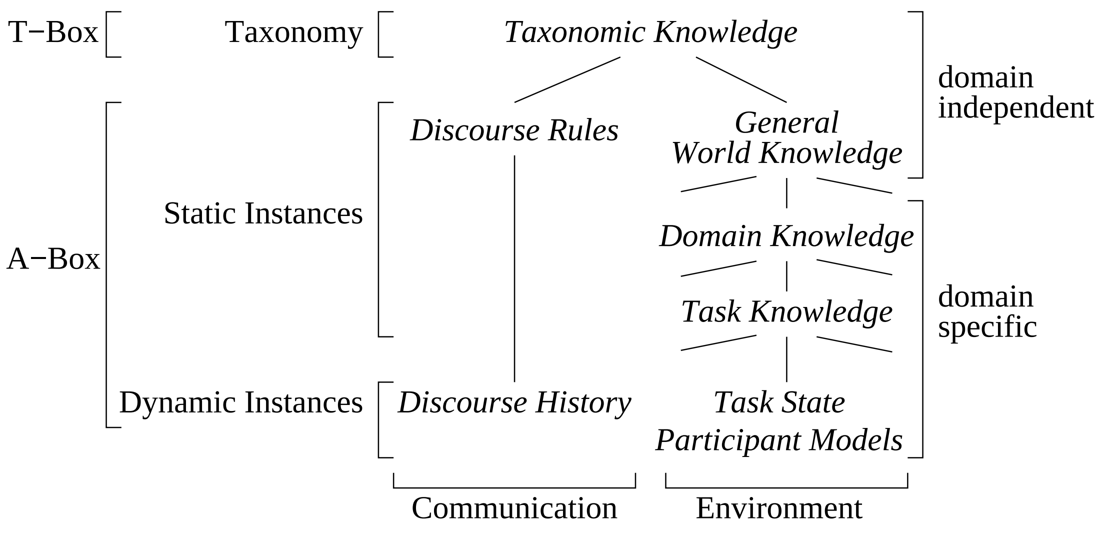
Level	**Primitives**	**Interpretation**	**Main Feature**
Logical	Predicates, functions	Arbitrary	Formalization
Epistemological	Structuring relations	Arbitrary	Structure
Ontological	Ontological relations	Constrained	Meaning
Conceptual	Conceptual relations	Subjective	Conceptualization
Linguistic	Linguistic terms	Subjective	Language dependency

Figure 4.1: Classification of KR formalisms on several levels according to the kind of primitives used (from (Guarino, 1995, page 10))

Knowledge representation needs to be concerned with the *ontological* level, which is about the nature of things and the state of the world, and the *epistemological* level, which is about how knowledge can be extracted from the state of the world. (Guarino, 1995) distinguishes the levels of knowledge representation shown in the table in Figure 4.1. He describes the characteristics of the different levels as follows: The *logical level* works with predicates and functions that are given a formal semantics that allows for their formal interpretation but involves, however, no commitment regarding their interpretation. The *epistemological level* adds structure in terms of generic concepts and roles. The *ontological level* explicitly specifies the concepts and roles by characterizing the meaning of the basic ontological categories, and the relations between them. The *conceptual level* assigns the primitives a definite cognitive interpretation, e. g., elementary actions or thematic roles. Finally, the *linguistic level* is concerned with the concrete linguistic units.

4.2.2 Types of Knowledge

The knowledge base for a dialogue system needs to cover different areas, and it is advantageous to clearly separate the knowledge for each area to facilitate their reuse (Flycht-Eriksson, 1999).



type	example
taxonomic knowledge	integers are numbers, there is a partition of subcategories into “animate” or “inanimate”
discourse rules	A question requires an answer
general world knowledge	Ballack is a football player
domain knowledge	Ballack was injured in the last game
task knowledge	Logging in to the application requires a password
discourse history	Dialogue participant P_1 greeted participant P_2 and then asked me about the weather
task state	User U has completed the login procedure
participant model	Dialogue participant P wants to sell me something

Figure 4.2: Knowledge types and examples

The two main knowledge types are *task knowledge* and *discourse knowledge*. We will make a somewhat more fine-grained distinction, as shown at the top of figure 4.2. In the table on the bottom, concrete examples are given for each type of knowledge.

The knowledge types differ as to whether they are taxonomical in nature, or whether they represent concrete object entities, called *instances*. In description logics, these types are often referred to as the *T-Box* (terminological box) that contains the definitions of concepts the knowledge base is about, and the *A-Box* (assertional box) that contains descriptions of actual entities, or instances of the concepts. For dialogue systems, it is also useful to make a further distinction between knowledge that is used to represent the aspects of *communication*, and knowledge representing the *environment* in which the interaction takes place. Instances of both kinds can be either static or dynamically created and mutable during an interaction. The areas covered by the knowledge types are the following:

- *Taxonomic knowledge* describes the taxonomic relations between concepts (types of objects). It is concerned with concept hierarchies (e. g., *is-a* relations) and relations between instances of concepts (e. g., *has-a* relations).
- *General World Knowledge* concerns factual information about the world, which includes objects, their attributes, and relations between them. It is extended by *domain knowledge* which is about the concrete domain of the application.

  For example, for the *VirtualHuman* system that is concerned with the domain of football, the world knowledge defines that there are concepts like persons and numbers. The domain knowledge includes the concept of football player (a specialization of the person concept) with has attributes such as the player's name, physical fitness, or nationality. It specifies that the *FootballPlayer* category is a subcategory of *Person* and has subcategories such as *Defender, Goalkeeper* and so on.[1] The knowledge base also contains objects that are concrete instances of the *FootballPlayer* concept.

  It is not strictly necessary to separate general world knowledge and specific domain knowledge. However, doing so makes it possible to share and re-use existing bodies of domain-independent general world knowledge across different applications (cf. section 4.3.3).
- *Task knowledge* connects the interaction and the general world and domain knowledge with respect to the goals of the system. In *VirtualHuman*, it encompasses what the different quiz games involve and how the characters go about realizing their roles. Task knowledge is in terms of goals and the actions that are necessary to achieve them, how different actions change the state of the system's goals, and what they mean for the character's future actions, plans, and behavior. Task knowledge can be seen as a special case of domain knowledge.
- Knowledge about the *discourse rules* states what form appropriate contributions in a given context can take on, and what their meaning is. Coherent interaction has rules, and the individual communicative acts have an effect; for example, a question will be *about* some subject (that should be known and identifiable from the world knowledge) and influence the state of the question's addressee in some way, e. g., in that she will be obligated to return an answer. The dialogue games that are introduced later in this thesis are instances of discourse rules.
- The *discourse history* models the content and structure of instances of actual dialogical interactions between the interlocutors, and relations between them (e. g., belonging to the same topic, or cross-utterance references). Information that is obtained in the course of the dialogue may also be accepted permanently by dialogue participants to become part of their world or domain knowledge. For example, if the dialogue system in one session learns the address of a user, it could use this information to update its persistent world knowledge, so it would not have to ask for it again in a subsequent session.

[1]When we are talking about a concrete concept from a knowledge base, we emphasize this by using the name as one word in italics, e. g., *"FootballPlayer"* vs "football player".

Exchange of knowledge items between dialogue participants constitutes *communication*, and the discourse history records the communication in the system.

In a multimodal system, contributions in different modalities are stored in the discourse history. As will be explained in Section 5.3.2, our model relies on the *FADE* engine (Pfleger, 2007) to store the discourse history.

- Parallel to the discourse history, the situation also includes information that belongs to a *task state*. This information is influenced by the discourse, but separate from it. It comprises information like, e. g., *"The football team on the field is complete"* (in *VirtualHuman*). The task state can also change due to, e. g., physical actions of the characters, or external events that may not have anything to do with the interaction per se.
- A user model provides a dialogue system with an *"explicit model of the user's beliefs, goals and plans"* (Wahlster and Kobsa, 1989). It is usually considered distinct from the discourse and domain model. In our approach, the human users do not hold a special position, but are on a par with the other dialogue participants; each participant is therefore represented by *participant models* held by the other interlocutors as (dynamic) parts of their overall domain model. Models for virtual characters are a special case of participant models; we refer to them as *character models* (cf. Section 7.2.5).

The basic taxonomic structure and a part of the instances concerning stable world knowledge (i. e., knowledge that does not change during the course of interactions) can, in principle, be independent of the domain. Rules of discourse can also be largely domain-independent, since discourses in different domains are structured similarly (exceptions for domains where special or additional rules may hold nonwithstanding, such as in a classroom situation). We assume the taxonomy and the rules of discourse to be static (although both could also be dynamic in a learning system). On the other hand, the knowledge about the task, the task state, and the actual history of the current discourse are domain-specific to a high degree.

4.2.3 Dialogue and Context

To provide those dialogue participants that are agents controlled by the computer with some understanding of the task, each one needs to have access to (at least parts of) the general world, domain, and task knowledge. It is possible that they have different versions of this information, resulting in differing private views or degrees of being informed. For example, in a scenario with characters that take on the roles of a "teacher" and a "pupil", their knowledge of the subject matter will have to be different almost by definition.

Each participant also needs to be able to access a record of the current task state, and a (possibly also private) dialogue history that contains the parts of the interaction the participant has followed. When making their own contributions to the discourse, participants can resort to the discourse rules to determine their options. The principal uses for the different types of knowledge are:

- representing communication in terms of communicative acts passed between the participants, and in the dialogue history,

- representing the state of the world and the task,
- providing rules for the communication between dialogue participants, and between the components of the system, and
- providing computational dialogue participants with a way to reason about the state of the world and determining courses of action.

To be able to meaningfully and purposefully engage in a dialogue, the participant agents need an account for the changes effected by the utterances, and a way in which they can access and act upon that context.

By *context* we mean a knowledge base of facts about the domain and the task state that is dynamically updated to reflect the current state in the interaction. Similar to *SmartKom*'s information state, the (private) domain and task state information is separated from the (public) interaction history, and both types of information are processed by separate modules (namely, the conversation manager and the *FADE* engine). Also like in *SmartKom*, dialogue games are used to specify the rules for well-formed interactions that have declarative and procedural aspects (cf. Section 2.4.3). The model we present separates them and includes the declarative aspects of the dialogue games in the contexts of the participants as facts.

A system of multiple autonomous participants calls for a set of separate contexts, since the knowledge base of the participants may be different from the start, and/or change independently. Dialogues are taking place in some kind of environment, which can be the actual physical reality, but also a simulated reality represented graphically as a "virtual environment" as in *VirtualHuman*. The state of this environment, which we call the *system context*, represents the actual, "objective" state of affairs as modeled by the dialogue system. Additionally, each virtual character has a private *character context*, which holds its individual view of the current state of affairs that may or may not conform to the system context, and in addition a private set of beliefs, desires, and intentions.

Actions that take place in the environment bring about changes in the contexts of all the dialogue participants that perceive them. The interconnection between the contexts of the dialogue participants and the environment context is illustrated in an example in Figure 4.3. It shows an instantiation of the left side of the interaction triangle from Figure 2.7 with one human user and two characters, Peter and Mary. When Peter utters an invitation to Mary for dinner, he triggers several different kinds of context changes. The dialogue history will record the (objective) fact that Peter made the invitation. The changes in the contexts of the participants depend on their attitudes. Assuming both characters adhere to "normal" social conventions, Peter's context will include a notion that after an acceptance of the invitation by Mary, he will be obliged to cook a meal. On the other hand, Mary now would be obliged to address Peter's invitation by accepting or declining it, and if she accepts, she takes on an obligation to visit Peter. The user (or any other additional participant), not being involved directly in the interaction, might just notice the fact that Peter invited Mary. It would also be possible that a third-party observer draws additional conclusions not shown in the figure (e. g., assuming that Peter likes Mary).

Utterances can affect the participant models that contain information about the other interlocutors; also, a character's interpretation of an utterance, as well as its actions, may be

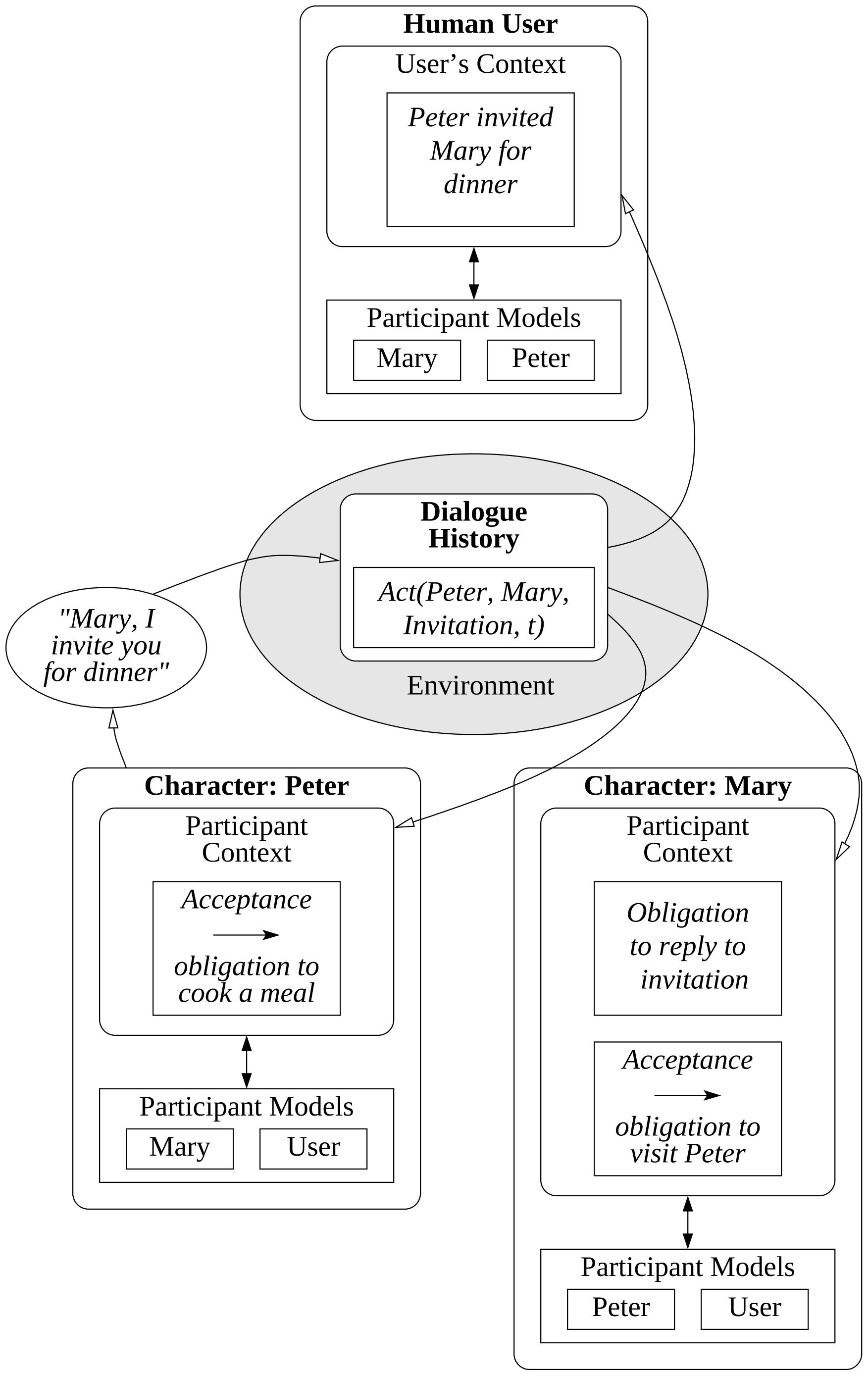

Human User
User's Context
Peter invited Mary for dinner
Participant Models
Mary
Peter
Dialogue History
Act(Peter, Mary, Invitation, t)
Environment
"Mary, I invite you for dinner"
Character: Peter
Participant Context
Acceptance → obligation to cook a meal
Participant Models
Mary
User
Character: Mary
Participant Context
Obligation to reply to invitation
Acceptance → obligation to visit Peter
Participant Models
Peter
User

Figure 4.3: Example for changes in different contexts effected by a dialogue act

influenced by its participant models (e. g., if it does not trust a dialogue partner). In the example, the invitation might cause a positive modification in Mary's model of Peter. Participant models generalize the notion of a *user model* employed in systems that involve interaction of a single human user with the computer for multi-party situations where models are required for both human users and virtual characters. The mental model of the human user is beyond the scope of this work; instead, we need to account for the system context and the contexts of the characters. The information in each of these contexts is described by means of a dedicated ontology, as we describe in the following sections.

4.3 Ontological Representation

4.3.1 Representation Formalisms

Modeling the structural and factual information about a given domain is a difficult and laborious task, and there is still no straightforward and universally accepted methodology for it. As we are also concerned with virtual environments and (multiple) characters, the domain for our system not only includes the conversation itself, but also the "physical" objects of the virtual environment, including the characters themselves, as well as models of the emotional and mental states of the characters. Still, an extensive section of the knowledge base is devoted to the representation of dialogue and dialogue structure. As outlined in chapter 3, dialogue systems use many different methods for knowledge representation. This section is about the reasons why we chose to use an ontological foundation for our framework.

A *declarative* representation of some content defines what it is about, in contrast to a *procedural* representation that implements directly how to manipulate and what to do with it. Using a declarative approach is increasingly considered good practice for knowledge-based systems, especially for larger domains, because modeling knowledge independently from the procedures that operate on it offers more flexibility than a procedural approach, allows for re-use of existing knowledge across systems and facilitates maintenance.

Knowledge can be represented in a wholly or partly declarative way by designing a *custom representation* tailored to the implementation requirements of a particular system or algorithm. For example, systems that implement a dialogue purely as state transitions in finite state automata frequently encode the knowledge in a custom FSA representation. While such an approach separates the knowledge from the implementation, the representation remains largely dependent on the particularities of the specific program, and a re-use for a different environment can turn out to be difficult or impossible.

An abstract, expressive and concise representation method is to encode the knowledge as assertions in some logic. Logic formalisms range from elementary propositional logic, over first-order logic, to higher-order logics and variants which allow capturing notions of, e. g., possibility, necessity, time, or belief (modal logics), and uncertainty (fuzzy logics). Such extensions can be powerful tools for dealing with various phenomena in natural language (Gamut, 1991), however, a problem with logics that go beyond propositional logic is that it is difficult to pose a limit on computational requirements: determining the truth value of assertions is not decidable in the general case (Fitting, 1990).

Computational ontologies provide a vocabulary to link assertions in some, possibly restricted, logic to define concepts (types of entities) and their instances. OWL, a language recommended by the World-Wide Web Consortium (W3C) for ontology specification, uses a *description logic*, which is a subset of first-order logic derived from frame-based systems, semantic networks, and KL-ONE-like languages (Nardi and Brachman, 2002). While description logics put some constraints on what can be expressed, they have the big advantage that there are decidable procedures to determine the truth value of expressions.

Gruber defines a (computational) ontology as follows:

> *"An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge.* [...] *Formally, an ontology is the statement of a logical theory."* (Gruber, 1993, page 1)

Although formal ontology-based domain modelling for dialogue systems has been used successfully for some time (see, e. g., (Porzel et al., 2003)), it has only recently begun to become standardized and feasible for larger domains with the introduction of languages such as *RDF(S)* and *OWL*[2]. Tools for ontology creation and editing have also begun to emerge, such as the *Protégé* and *OilEd* visual editors, and program libraries to access and reason with ontologies, like the *Jena* API.[3] These efforts related to making a semantic web access feasible have resulted in the adoption of a set of standards for ontology description that have gained widespread acceptance. A third strong incentive to use ontologies comes from the possibility to re-use predefined concepts from *upper* or *base ontologies*, such as *SUMO*, *DOLCE*, or *OpenCyc* as a foundation upon which application-specific ontologies can be constructed (A synopsis of seven base ontologies is given in (Mascardi et al., 2007)).

Dialogue research begins to exploit the advantages of upper-level ontologies where they are available to support all areas of dialogue processing, also including recognition, interpretation, and generation. Other efforts create, use or merge general purpose ontologies, especially ones concerned with information retrieval in the Semantic Web. For example, for the *SmartWeb* project, *DOLCE* and *SUMO* were integrated to form the *SmartSUMO* foundational ontology, on which the *SmartWeb Integrated Ontology (SWIntO)* is based (Oberle et al., 2007).

4.3.2 Basic Ontological and Epistemological Terminology

An ontology imposes a taxonomy that divides the domain into different hierarchically ordered classes to represent types of entities, called *concepts* or *types*. For example, a *Tiger* concept would represent all kinds of tigers in the domain. The hierarchy defines *subconcept* and *superconcept* relations. The transitive subconcept relation is called the *is-a* relation. Actual objects in the domain that are described by a concept are called *instances* of that concept and

[2] see *http://www.w3.org/TR/rdf-schema/* and *http://www.w3.org/2004/OWL/*
[3] see *http://jena.sourceforge.net/*

its superconcepts: the individual tigers in the domain would be instances of the *Tiger* concept. Categories can be *concrete* or *abstract*. Categories that are used to represent actual entities in the domain are called *concrete* categories, in contrast to *abstract* categories that only have instances that also belong to at least one concrete subcategory. For example, the ontology can have an abstract *Animal* concept as a superconcept of the concrete animal concepts *Tiger*, *Elephant*, and so on.

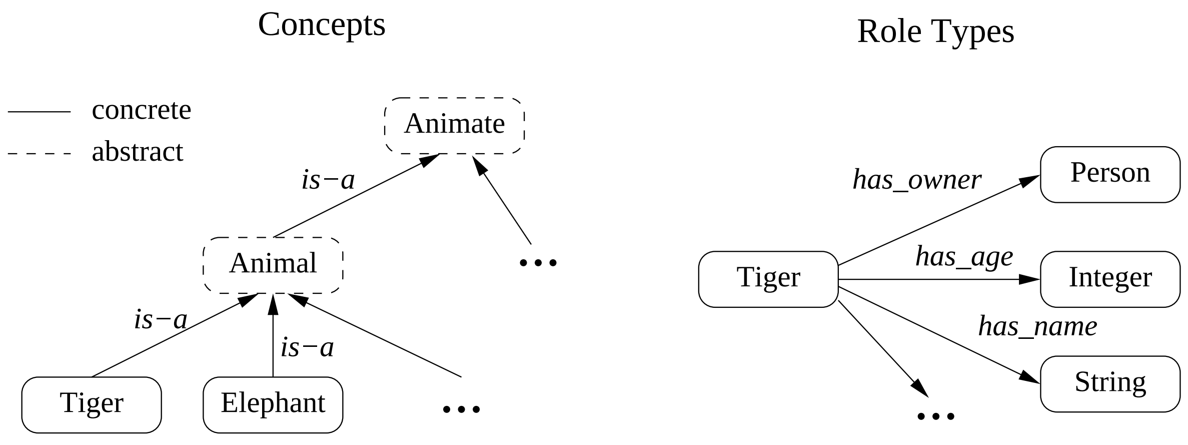


Figure 4.4: Illustration of some ontological and epistemological terms

On the epistemological level, a concept can define *role* relations that relate its instances to other instances of a given type; e. g., "*has-a*" relations. Roles describe the attributes of the instances of a concept. For example, the *Tiger* concept could have a role *has_owner* of type *Person*, which means that a *Tiger* instance is associated with an *Person* instance that represents the owner of the Tiger, or that a *Tiger* "has-an" owner.[4] We call the instance associated with the role *r* of an instance via the relation *has_r* the *value* or *role filler* of the role *r*. Instances of subconcepts of the role type are also possible role fillers.[5]

The cardinality of values for a given role of a *valid* instance (an instance that is considered to be completely specified—see section 4.4.2 on underspecification) can be restricted. If more than a single value is allowed for a role (e. g., a *Tiger* could have more than one owner), the role is called *multiple*. If there must be at least one value of a role for an instance to be valid, the role is called *required*; otherwise it is *optional*.

Subconcepts inherit the roles from their superconcepts and can add new roles of their own, thereby *extending* their superconcepts. A concept *Person* that is a subconcept of an *Animal* concept inherits, e. g., the *has_age* role, and adds new person-specific roles, such as *has_nationality*. Subconcepts can also further restrict the concepts that are allowed for role values; the *Person* concept will restrict the *has_parent* slot that it inherits from *Animal* to have only values of type *Person* instead of *Animal*. In the case of *multiple inheritance*, a concept can have multiple superconcepts and inherit roles from all of them. Multiple inheritance can

[4]We adopt the convention that a "possession" of an attribute *a* is expressed with a role *has_a*, whereas a "predicate" role, which is filled by a boolean value indicating whether *a* is the case is named *is_a*.

[5]In addition to concepts explicitly defined in the hierarchy, a role can also specify one of a small set of atomic types for the role fillers that correspond to the simple datatypes used in many programming languages, e. g., *String*, *Integer*, or *Boolean*.

be difficult to maintain in larger ontologies because it can lead to contradictions (Bechhofer et al., 2002). The systems described in this thesis did not require multiple inheritance.[6]

4.3.3 Defining the Base Ontology

For ontological domain modeling, (Bateman, 1990) argues that the linguistic knowledge should be factored out from task knowledge to obtain a so-called *upper model* that is reusable and domain-independent. An upper model can also be obtained from adapting a *base ontology* that describes foundational concepts that are general enough to be applicable in a wide range of domains, and is designed to be extendable to meet the additional requirements introduced by a specific domain.

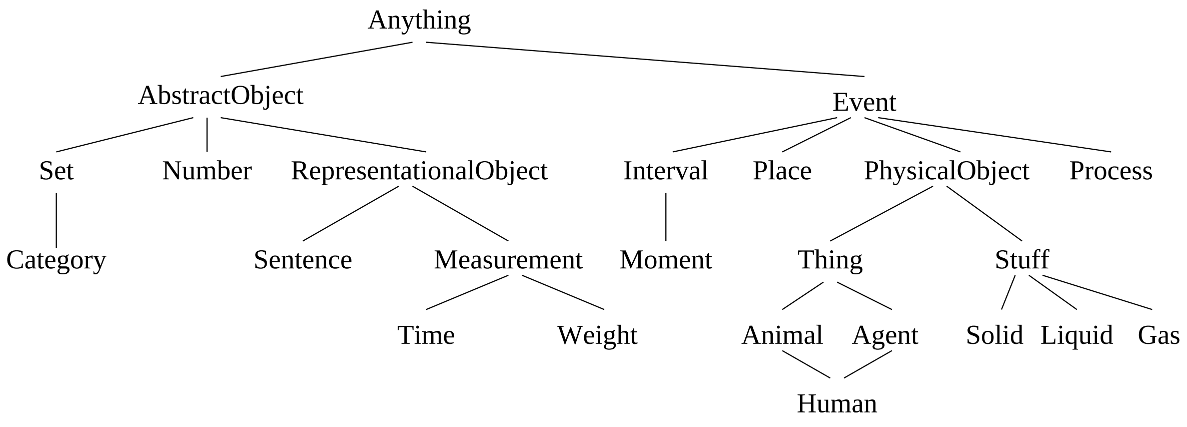


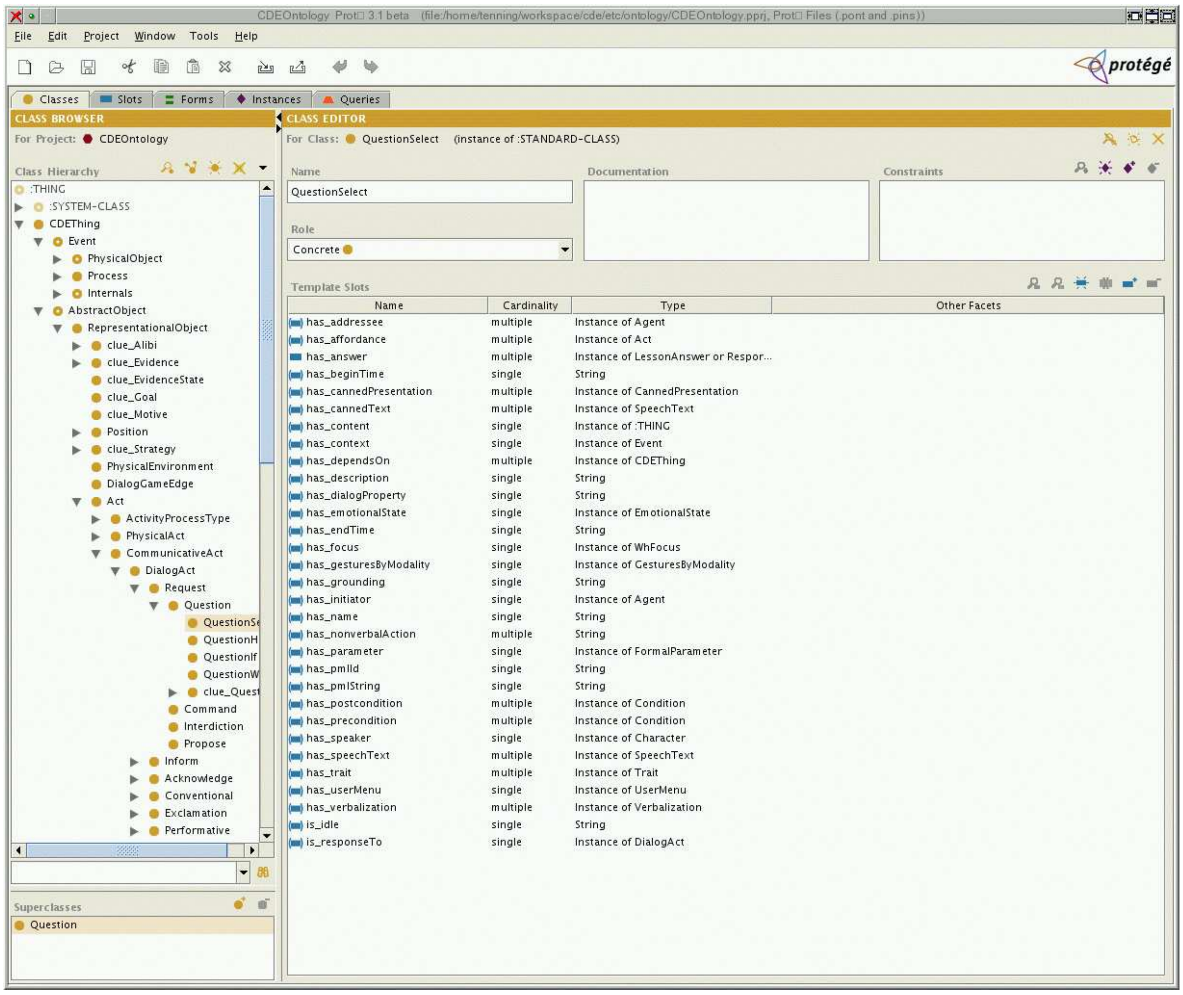
Figure 4.5: Base ontology structure

There are considerable efforts to examine the process of designing domain ontologies for, e. g., Semantic Web content (Berners-Lee et al., 2003), but also notably for applications in the fields of medicine and biology (e. g., using medical ontologies to help diagnose illnesses (Milward and Beveridge, 2003)) . Figure 4.5 shows the top-level skeleton of the base ontology of (Russell and Norvig, 1995), containing some initial sub-divisions along abstract categories. As can be seen from the figure, the ontology also includes important auxiliary concepts that could be positioned between structural and world knowledge in the types of knowledge listed in section 4.2.2, e. g., the concepts modeling *Set*s, *Number*s, and *Time*. This base structure is used as a starting point for the domains of the *VirtualHuman* and *Clue* systems. For the *OMDIP* system, the base ontology from *SmartWeb* (Sonntag and Romanelli, 2006) is adapted.

As an editor to develop and maintain the knowledge base, we use *Protégé*, a free, open-source tool for creation and visualization of ontologies in a graphical environment (Gennari et al., 2003).[7] *Protégé* allows the knowledge engineer to edit the taxonomic hierarchy and concept instances separately. Figure 4.6 shows the *Protégé* interface for defining concept hierarchies and the slots for different concepts. While *Protégé* can be very helpful for designing taxonomic hierarchies, the definition of instances can quickly become cumbersome, since there is no easy

[6]The *JenaLite* tool implemented for our framework (see Section 6.2.2) also does not currently support multiple inheritance for this reason.

[7]*Protégé* website: *http://protege.stanford.edu*


CDEOntology Prot□ 3.1 beta (file:/home/tenning/workspace/cde/etc/ontology/CDEOntology.pprj, Prot□ Files (.pont and .pins))
File Edit Project Window Tools Help
protégé
Classes Slots Forms Instances Queries
CLASS BROWSER
CLASS EDITOR

For Project: CDEOntology

Class Hierarchy

```
:THING
▶ :SYSTEM-CLASS
▼ CDEThing
  ▼ Event
    ▶ PhysicalObject
    ▶ Process
    ▶ Internals
  ▼ AbstractObject
    ▼ RepresentationalObject
      ▶ clue_Alibi
      ▶ clue_Evidence
        clue_EvidenceState
        clue_Goal
        clue_Motive
      ▶ Position
      ▶ clue_Strategy
        PhysicalEnvironment
        DialogGameEdge
    ▼ Act
      ▶ ActivityProcessType
      ▶ PhysicalAct
      ▼ CommunicativeAct
        ▼ DialogAct
          ▼ Request
            ▼ Question
                QuestionS
                QuestionH
                QuestionIf
                QuestionW
              ▶ clue_Quest
              Command
              Interdiction
              Propose
          ▶ Inform
          ▶ Acknowledge
          ▶ Conventional
          ▶ Exclamation
          ▶ Performative
```

Superclasses: Question

For Class: QuestionSelect (instance of :STANDARD-CLASS)

Name: QuestionSelect

Role: Concrete

Documentation

Constraints

Template Slots

Name	Cardinality	Type	Other Facets
has_addressee	multiple	Instance of Agent	
has_affordance	multiple	Instance of Act	
has_answer	multiple	Instance of LessonAnswer or Respor...	
has_beginTime	single	String	
has_cannedPresentation	multiple	Instance of CannedPresentation	
has_cannedText	multiple	Instance of SpeechText	
has_content	single	Instance of :THING	
has_context	single	Instance of Event	
has_dependsOn	multiple	Instance of CDEThing	
has_description	single	String	
has_dialogProperty	single	String	
has_emotionalState	single	Instance of EmotionalState	
has_endTime	single	String	
has_focus	single	Instance of WhFocus	
has_gesturesByModality	single	Instance of GesturesByModality	
has_grounding	single	String	
has_initiator	single	Instance of Agent	
has_name	single	String	
has_nonverbalAction	multiple	String	
has_parameter	single	Instance of FormalParameter	
has_pmlId	single	String	
has_pmlString	single	String	
has_postcondition	multiple	Instance of Condition	
has_precondition	multiple	Instance of Condition	
has_speaker	single	Instance of Character	
has_speechText	multiple	Instance of SpeechText	
has_trait	multiple	Instance of Trait	
has_userMenu	single	Instance of UserMenu	
has_verbalization	multiple	Instance of Verbalization	
is_idle	single	String	
is_responseTo	single	Instance of DialogAct	

Figure 4.6: Editing ontological concept definitions in *Protégé*

way to track all instances. The native storage format of the *Protégé* editor is not a standardized one, and frequently changes while the editor is still under development.

Ontologies developed with *Protégé* can also be stored, among other formats, as standardized *RDF* (Resource Description Framework) resources structured by *RDFS* (RDF Schema). *RDFS* was designed as a general language to describe the vocabulary for RDF in a particular domain, and can be used to represent ontologies. It encodes the subconcept hierarchies and the role value types and ranges. The resources themselves are a collection of *RDF triples*, each of which specifies a subject-predicate-object relation, such as *(Elephant, is-a, Animal)* or *(Tiger*$_{34}$*, has_owner, Person*$_{3}$*)*.

4.3.4 Mapping to an XML representation

In our framework, the RDF(S) representation is not used directly for performance reasons (see Section 6.2.2). Instead, as was already done in the *SmartKom* and *SmartWeb* systems, an additional layer is employed that expresses the content in terms of an XML representation based on typed feature structures (TFS) (Carpenter, 1992). The original ontology, which can be present in either RDF(S) format (or alternatively the proprietary ontology storage format of the *Protégé* editor) is translated by a preprocessor into the format that is easier to manipulate and also (arguably) more human-readable. An excerpt of the *VirtualHuman* ontology file rendered in this format is shown in figure 4.7 (see Appendix E for an explanation of the abbreviated XML notation). The XML representation comprises two sections:

- **Concepts section**

  Here, all concepts are defined, and for each concept the set of its (immediate) superconcepts is given. Each concept entry lists all roles defined by the concept along with their restrictions. Roles inherited by superconcepts are not listed again. The role definitions state whether the role is required and/or multiple.[8] The *"type"* attribute defines whether the role value fillers must be ontological instances or members of the atomic datatypes available in *Protégé*. In case of *Instance* role values, an additional attribute lists the allowed concepts legal filler instances may belong to.

- **Instances section**

  This section defines all instances, assigns them a unique identifier, and specifies their (possibly multiple) role values. As can be seen in Figure 4.7, this also includes the roles that come from superconcepts (the *Midfielder* instance also has slots inherited from *Person* and *FootballPlayer*). Values of an atomic datatype can be specified directly as a string literal, whereas an instance value is referenced by its identifier; in this case, another unique instance with that identifier must also be be defined in the same file.

  Instances are not required to be *totally well-typed*, i. e., even if the concept defines a role to be required, it is still possible to define an instance without a value for it. This instance will then be underspecified. The intended interpretation is that the information is necessary for a full specification, but not given: e. g., the ontology may acknowledge that every *FootballPlayer* has a first name, but not include this information for a certain player instance.

This representation is used to construct a type hierarchy and corresponding instances like in the *SmartKom* system (Alexandersson and Becker, 2001; Gurevych et al., 2006). The translation process is essentially straigthforward: the concept hierarchy is mapped to a type hierarchy, roles to corresponding slots, and ontological instances to instances of XML-TFS structures.[9]

Figure 4.8 shows an example of the TFS notation for an instance of the *Menu* concept. In this notation, we use a star postfix ($*$) to indicate that a feature may be multi-valued. If there is more than one value present for such a feature, we show them as lists in angle brackets ($<>$). Value types are shown in slanted font (*Menu*). Given atomic values are quoted (*"please select"*), while values that are TFS themselves are in square brackets. Where an actual value

[8]The possibility to specify exact numerical cardinalities for slots is not supported in the current format, but could easily be added.

[9]Some aspects, e. g., exact cardinality restrictions for slots, are left out in our translation since they were not required for the applications described in this thesis; however, in case they they were needed for a future application, the translation could be adapted without too much effort.

```
ontology
  concepts
    FootballPlayer
      superconcepts
        Person
      roles
        is_listed type="String" multiple="no" required="no"
        has_fitness type="String" multiple="no" required="no"
        has_speed type="Integer" multiple="no" required="no"
        has_footballAction type=Instance" multiple="no" required="no"
                          allowedConcepts="FootballAction"
        ...
    Midfielder
      superconcepts
        FootballPlayer
        ...
    FootballAction
      superconcepts
        MotionProcess
      roles
        has_style type="Instance" multiple="no" required="no"
                  allowedConcepts="CDEThing"
        has_agent type="Instance" multiple="no" required="no"
                  allowedConcepts="Agent"
        has_direction type="Instance" multiple="no" required="no"
                      allowedConcepts="Direction"
        ...
    Injury
      superconcepts
        FootballAction
      superconcepts
        roles
          has_cause type="Instance" multiple="no" required="no"
                    allowedConcepts="MotionProcess"
          has_partOfTheBody type="Instance" multiple="no" required="no"
                            allowedConcepts="BodyPart"
          ...
  instances
    Midfielder id="CDEOntology_Instance_470000"
      has_firstName: Michael
      has_lastName: Ballack
      is_listed: true
      has_fitness: 94
      has_footballAction id="CDEOntology_Instance_730000"
      ...
```

Figure 4.7: Excerpt from the XML ontology representation in *VirtualHuman*

of an instance is not important or not known, we also sometimes just give the type of a value (like in the case of the HAS_NAME slot).

$$\begin{bmatrix} \textit{Menu} & \\ \textsc{has\_name} & \textit{String} \\ \textsc{has\_title} & \textit{“please select”} \\ \textsc{has\_menuItemList*} & \left\langle \begin{bmatrix} \textit{MenuItem} & \\ \textsc{has\_label} & \textit{“Yes”} \end{bmatrix}, \begin{bmatrix} \textit{MenuItem} & \\ \textsc{has\_label} & \textit{“No”} \end{bmatrix} \right\rangle \end{bmatrix}$$

Figure 4.8: A *Menu* object represented in TFS notation

We call the result of the XML transformation, comprising a set of concept definitions and instances of these concepts, an *XML-Ontology*. We denotate the set of all instances of a concept C in an XML-Ontology Ω (including instances of subconcepts of C) as $i_C(\Omega)$, and the set of all possible instances of C as $i^*_C(\Omega)$. Two XML-Ontologies are called *compatible* if all concepts that are present in both have the same set of slots, and there are no conflicts in the subtype hierarchy. In the following, we use depictions as in Figure 4.8 for typed feature structures of the corresponding XML-Ontology as convenient representations for ontological concepts and instances.

4.3.5 Using the XML-Ontology as a Data Structure

During the interaction, the ontology must be dynamically accessible and mutable as a data structure. As each character has a private ontology, there must be separate instances that will hold different information. Nevertheless, all instances must be *compatible* in the sense that expressions in one ontology must be interpretable in the others, otherwise mutual understanding would not be possible. One approach would be to apply an *ontology transformation* process to convert instances between ontologies; however, devising such a transformation for arbitrary pairs of ontologies is a difficult problem and currently not feasible without human intervention (McDermott et al., 2002); it has even been called *“certainly AI-complete”* (Dou et al., 2003, page 957)

Fortunately, in our case, it can be assumed that the knowledge bases of the characters are not fundamentally different, since we only want to model, differing knowledge on part of, e. g., a quiz show moderator and an football expert in the same scenario. We require that the taxonomic structure of all ontologies employed in the same system are compatible in the sense described in the previous section. On the other hand, the sets of concrete entity instances may differ and are indeed subject to change during the dialogue.

The initial ontologies, which are initialized from an external data source for each character at the beginning, contains its a priori world, domain, and task knowledge and the discourse rules as static entities. The discourse contributions and the task state are dynamic and can be changed. The domain ontology is an adaptation and extension of a base ontology, as described in the last section. For the extension, we use a dedicated namespace to integrate concepts that are specific to the workings of the model; concepts for domain description can be added in any namespace or reused if they are already present in the base ontology.

To support our dialogue management model, a collection of concepts inheriting from a *RepresentationalObject* concept was introduced, which we call the “dialogue management branch”

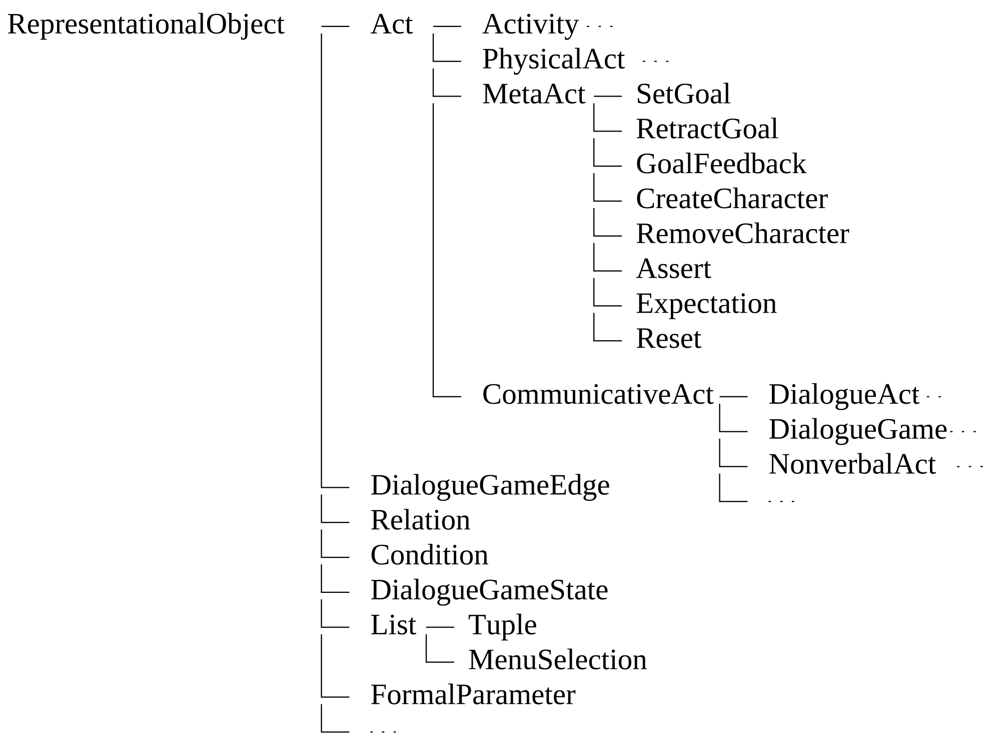


Figure 4.9: Some newly introduced subconcepts of *RepresentationalObject*

of the ontology. Figure 4.9 shows some immediate concept descendants in the subconcept tree of *RepresentationalObject*. Some of the base concepts, e. g., *Relation*, *List*, or *Tuple*, are auxiliary concepts needed for domain-independent operations over ontological objects. They are further extended by additional domain-specific concepts that are introduced by the task-specific part of the ontology, e. g., in *VirtualHuman*, the *MenuSelection* concept, which represents a selection list which is to be displayed graphically and includes additional attributes, e. g., screen position information. Besides the auxiliary classes, the bulk of the newly introduced concepts are related to the representation of dialogue acts, dialogue games, and expressing logical conditions.[10]

Objects the character "knows about" are present in the ontology as instances. The character can create new instances or modify existing ones while it learns new information about the environment. For example, the ontology of the expert character *Herzog* in *VirtualHuman* contains the following *Midfielder* instance:

Midfielder	
HAS_FIRSTNAME	*"Michael"*
HAS_LASTNAME	*"Ballack"*
IS_LISTED	*"true"*
HAS_FITNESS	*94*
. . .	

[10]Some additional subconcepts of the *Act* concept are shown in the chapter on the CDE model, see Figure 5.9 on page 140; all *CommunicativeAct* instances are listed in Appendix D).

The character therefore has information that there is a midfielder named Michael Ballack that is listed (in the national team roster), and has a quantitative assessment of his fitness. The knowledge base of the moderator, who does not have an opinion about Michael Ballack's fitness, does not need to contain detailed information about his fitness etc.; the moderator could also represent Ballack as a *FootballPlayer* instance instead of a more specific *Midfielder*.

The distinction between *descriptions* of objects and the objects per se is important. If dialogue participants talk about objects, they always exchange descriptions. To make an utterance about some object, the initiator needs to create a description of a mental object, i. e., *encode* it in some representation. This involves a selection of which aspects to include into the utterance. Utterances normally will not include everything the initiator knows about the object, but be restricted to the *relevant* aspects (Grice, 1975). For example, if one wants to inform somebody about a meeting with a joint acquaintance, it is sufficient to identify the name, as in *"I met John yesterday"*, in contrast to *"I met John Barry yesterday, who is 39 years old, and works in the bakery ... "* and so on. When the relevant parts of the ontological representation have been determined and encoded, they must then be *serialized* to fit in a linear *message* that can be communicated. This is akin to serializing an utterance as a sequence of phonemes when making a spoken statement.

On the opposite side, to be able to understand a contribution by another dialogue participant in the form of an encoded message, a listener needs to decode the message and relate its contents to its private ontological representation. Only then will he be able to integrate it into its own private view of the world. This process establishes the *common ground* reached by the utterance and is known as *grounding* the message (see section 4.4.3). Depending on the ontology of the recipient, it is possible that the content of the message will *mean* something different in the framework of the recipient's knowledge. To prevent too many misunderstandings, the knowledge bases of both communication participants need to be sufficiently similar, so that knowledge items from one can be mapped to equivalent items in the other. This is the case for compatible ontologies.[11]

4.4 Important Methods and Concepts

4.4.1 Unification, Restricted Unification and Overlay

The *unification* operation (denotated by the symbol $\sqcup$) is a standard operation that has been defined for logic, TFS, and other knowledge representation expressions. The operation can be used to merge two knowledge structures to produce a new, unified result that contains the combined information of both arguments (Carpenter, 1992). It can fail if the structures cannot be reconciled because they are of incompatible types, or contain incompatible information. The result of unification is the greatest lower bound of both arguments, or failure if there is no such lower bound.

Figure 4.10 shows an example for the unification of two feature structures of type A and B, where A is a supertype of B. If, in this example, the feature ALPHA had a different value in

[11] Strictly speaking, it would be sufficient to require that parts of the ontology that hold information *intended to be communicated* are compatible, but not ones that are, e. g., used internally by a character to make inferences.

$$\begin{bmatrix} A & \\ \text{ALPHA} & 1 \\ \text{BETA} & 2 \\ \text{GAMMA} & \begin{bmatrix} C & \\ \text{DELTA} & 3 \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} B & \\ \text{ALPHA} & 1 \\ \text{GAMMA} & \begin{bmatrix} C & \\ \text{EPSILON} & 4 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} B & \\ \text{ALPHA} & 1 \\ \text{BETA} & 2 \\ \text{GAMMA} & \begin{bmatrix} C & \\ \text{DELTA} & 3 \\ \text{EPSILON} & 4 \end{bmatrix} \end{bmatrix}$$

Figure 4.10: An example for unification

one of the arguments (a *value clash*), the unification would fail, and also if A and B were not in a (transitive) subtype relation (a *type clash*).

Restricted Unification, a variation of unification used for pattern matching, additionally imposes an ordering on the arguments and requires that all information in the first argument is also present in the second argument, otherwise it fails. The effect is that the result is either failure or the first argument strictly extended by the information in the second argument. A restricted unification for the arguments in the example figure would fail, since the second argument does not include any suitable values for the BETA and DELTA slots.

Overlay is an operation that is based on unification and can be used to integrate newly acquired knowledge with previous knowledge. The new information, called the "cover" is imposed over the "background" consisting of old information. Unlike unification, overlay is an operation that never fails, even when the covering and the background happen to have value or type clashes. The result of the overlay operation is a combined structure together with an *overlay score* that expresses how well the covering fits the background. Overlay is used in dialogue management to integrate new information (e. g., from a current utterance) with previous background information (e. g., from the dialogue history). The new information supersedes the old while preserving as much of it as possible.

The unification and overlay operations are defined formally in (Pfleger et al., 2002; Alexandersson and Becker, 2001). The working of the overlay of two structures *covering* and *background* can be briefly summarized as follows:

- it behaves identically to unification when there are no type clashes.
- if there is a type clash between a feature c from the *covering* and a feature b from the *background*,
  - if the features are atomic, the value from c (new information) is kept in the result; i. e., the *background* is overwritten.
  - if the features are complex, the types of the values of c and b are generalized to the least upper bound of *type*(c) and *type*(b) and overlay is applied recursively.

The score for the result is computed using the following parameters:

- *Covering (co)*: incremented for each case a TFS or an atomic value stemming from the covering is added to the result.

- *Background (bg)*: incremented for each TFS or an atomic value in the result occurs in the background.
- *Type clash (tc)*: incremented for each case where the type of the covering and the background was not identical.
- *Conflicting values (cv)*: incremented for each case where the value of a feature from the background is overwritten.

The formula to compute the overlay score is

$$score(co, bg, tc, cv) = \frac{co + bg - (tc + cv)}{co + bg + (tc - cv)}$$

An augmentation of the overlay operation was presented in (Alexandersson et al., 2004b). There, the type clash score (*tc*) is weighted by taking into account the *informational distance* of its arguments, i. e., how far apart the types of the overlayed instances are in the type hierarchy.

4.4.2 Underspecification and Matching

If the attributes of an entity are not fully known, or possibly even the exact mapping to a concept is unclear or ambiguous (e. g., it is known that the instance *MichaelBallack* refers to a *Person*, but given the state of knowledge, the more rigid subclass *FootballPlayer* cannot (yet) be inferred), it is common practice in dialogue systems and knowledge representation frameworks to use an *underspecified* representation. An underspecified instance can be interpreted to represent the set of all instances that unify with it.

$$\begin{bmatrix} \textsf{\textit{FootballPlayer}} & \\ \textsc{has\_nationality} & \textit{“German”} \\ \textsc{is\_listed} & \textit{true} \end{bmatrix}$$

Figure 4.11: An underspecified *FootballPlayer* instance

For example, Figure 4.11 shows a TFS instance that can be taken to represent all football players that are listed in the German national team. An additional reason to use an underspecified entity is that in some cases a partial description is sufficient to identify it by matching it with the elements of a set of possible candidates, and efficiency can be improved by reducing the amount of information that has to be communicated (Pinkal, 1999).

To determine if a “foreground” TFS matches another underspecified “background” TFS, it can be compared to and merged with the background knowledge by the unification operation. We call an underspecified instance intended to be matched against other instances an *instance template* or simply a *template*. To select one of a set of instance templates that matches a given instance most closely, we use a function called *best match* that is based on the overlay operation and also takes the informational distance of object instances into account (see section 4.4.1).

There are many ways to specify what constitutes a "good" match; one possibility is to measure it using the overlay score. Given an instance i and a set of instance templates T, the best match function selects the element $m \in T$ that gives the highest score:

$$\mathit{bestMatch}(i,T) = \arg\max_{m \in T}(\mathit{overlayScore}(i,m))$$

Note that it is possible that more than one score is maximal, in this case, the result is a set of instances that match e equally well. A *strict* best match additionally requires that all the features present in the background are present in the covering, i. e., the covering strictly extends the background (in this case, there must be no conflicting values, i. e., the *cv* parameter must always be zero). It is possible that no strict best match exists.

4.4.3 Grounding

Participants in a dialogue need to have knowledge about the world, and dialogue involves the passing of information between the participants. The information contained in dialogue acts needs to be derived from the knowledge base of the act initiator, and to be integrated into the knowledge base of the addressee(s). The information in the dialogue act itself is *detached* from the knowledge base it comes from, and a recipient needs to determine how to fit it into its own system of beliefs and thoughts, a process known as the *grounding* of information. It may be that the utterance itself relies on shared information to be meaningful, as is the case with e. g. elliptical utterances.

A different case is that the belief system of the interlocutors have a differing taxonomical structure. An example of this is that, e. g., a car mechanic talking about cars could have a different, more intricate model of a car than a listening layperson with a more simplistic model, and the listener therefore might not be able to understand an utterance from the mechanic with their own knowledge, either because concepts are missing, or the structure itself is different. Our model does not deal with this interesting problem, but requires that the ontologies of communicating agents have compatible concept hierarchies.

Figure 4.12 shows an example. One of the dialogue participants, a football expert, communicates the fact that *"Michael Ballack prefers to play in the midfield on the left side"* to the moderator, who is a layperson. The fact is derived from the expert's "professional" knowledge about football players, which also includes many other pieces of information about Michael Ballack that are not related to the statement, such as his physical fitness. Following the Gricean Maxims, the expert constructs the utterance so that it only includes the relevant information. In this case, even the TFS type is generalized to *FootballPlayer* instead of the more specific *Midfielder*. The moderator then tries to integrate the content into his ontology. He has far inferior knowledge, in the example he knows only the last names of the players. He also has incorrect information and believes that Ballack is not listed in the team roster. Overlaying the new content on the best match results in the TFS at the bottom, which includes the new information, which replaces the previous belief.

Unfortunately in this situation, the information about the listing status was not included in the utterance (because the expert did not deem it relevant), therefore, the moderator continues to hold on to his false belief. If it *had been* included, the moderator would not

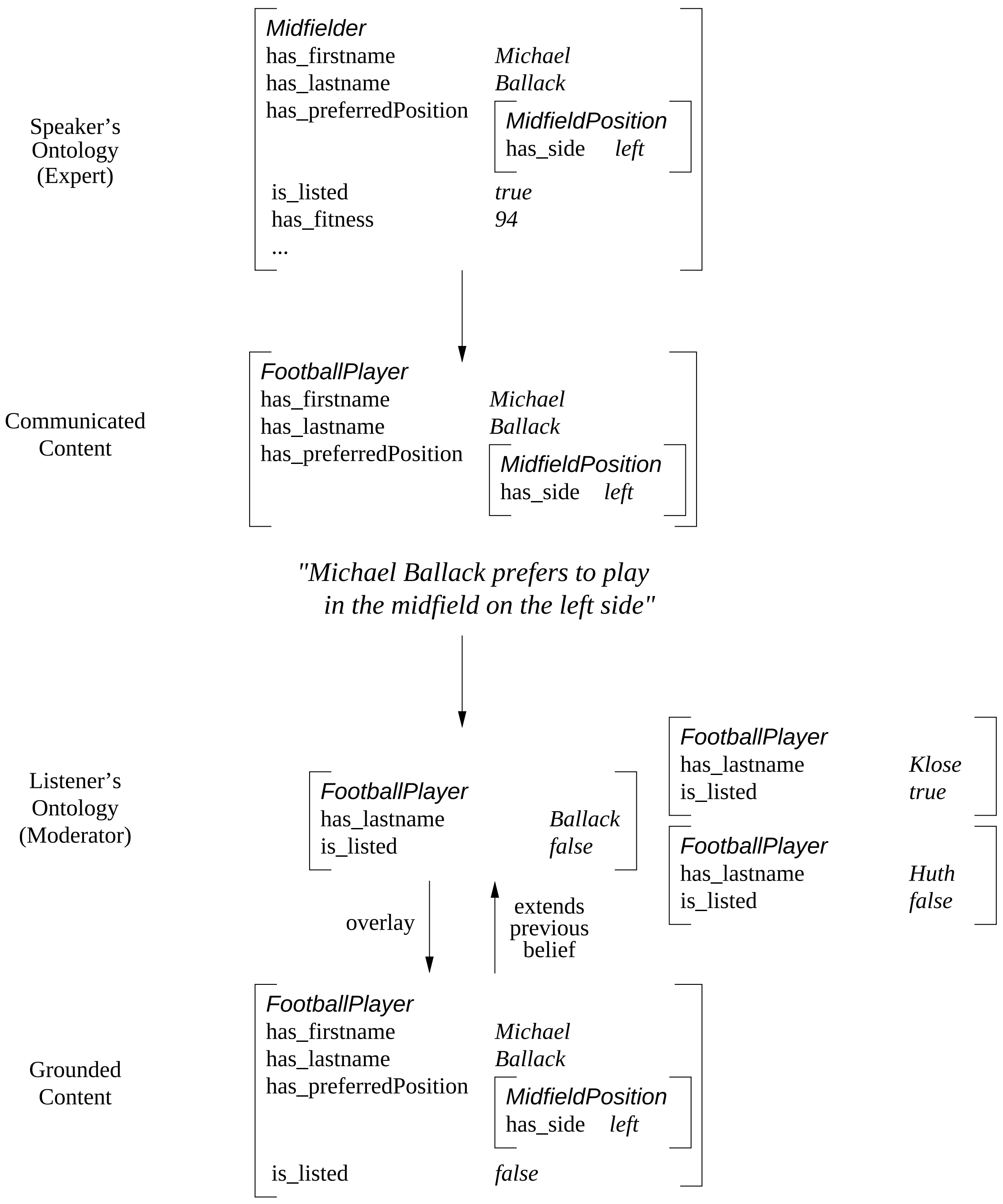


Figure 4.12: Example for the grounding process

have found a best match (the TFS representing *Ballack* and *Klose* would have had the same number of clashes, namely, one). There are several possibilities to deal with this complication and rectify the false belief of the moderator. First, the tie of scores would not occur if the moderator had more knowledge about the players (e. g., if he also knew the first names). Second, the slots of the TFS could be assigned *weights* in the ontology itself to account for their relative significance, assigning the *has_name* slot a greater weight than the *is_listed* slot. If both resolution possibilities are not available, the moderator can do nothing but ask a clarification question to find out who the expert was referring to.

4.4.4 Relations and Condition Matching

The presence of and object instance in the ontology of a dialogue participant means that the participant believes that the object exists, and has the attributes of the instance. A special kind in this regard are instances of the type *Relation*. Such instances assert that a named relation holds between a tuple of other instances.

$$\begin{bmatrix} \textit{Relation} & \\ \textsc{has\_relationName} & \textit{holds} \\ \textsc{has\_tupleSchema} & \left\langle \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{1} \\ \textsc{has\_name} & \textit{person} \end{bmatrix}, \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{2} \\ \textsc{has\_name} & \textit{object} \end{bmatrix} \right\rangle \\ \textsc{has\_tuple} & \begin{bmatrix} \textit{Tuple} & \\ \textsc{has\_arity} & \textit{2} \\ \textsc{has\_tupleElement} & \left\langle \begin{matrix} \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{1} \\ \textsc{has\_value} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{MissPoisonella} \end{bmatrix} \end{bmatrix}, \\ \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{2} \\ \textsc{has\_value} & \begin{bmatrix} \textit{MurderWeapon} & \\ \textsc{has\_name} & \textit{knife} \end{bmatrix} \end{bmatrix} \end{matrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

Figure 4.13: A *Relation* instance

Figure 4.13 shows an example from the *Clue* system. This instance asserts that there is a relation *holds* between an instance representing a virtual character called *MissPoisonella* and an instance of the type *MurderWeapon*, a knife; in other words, that Miss Poisonella is in possession of the knife. There can be arbitrary many *Relation* instances for a given relation. Note that the *Tuple* concept is constructed in such a way as to allow the ordering of tuple elements by indices. This is necessary because ordered lists are not straightforward to model as TFS (cf. also (Romanelli, 2005). Another possibility would be to use *cons*-Lists as in the *Lisp* language, which, however, leads to a quite intricate representation. Another advantage of the *Tuple* representation is that it is easily possible to underspecify tuples by leaving out tuple elements. The optional *TupleSchema* slot, if present, makes it possible to assign names to the different positions in the tuple.

$$\begin{bmatrix}
\textit{Condition} & \\
\textsc{has\_argument} & \begin{bmatrix}\textit{FormalParameter} & \\ \textsc{has\_name} & A\end{bmatrix}\\
\textsc{has\_restriction} & \begin{bmatrix} \textit{Relation} & \\ \textsc{has\_relationName} & \textit{holds}\\ \textsc{has\_tupleSchema} & \left\langle \begin{bmatrix}\textit{TupleElement} & \\ \textsc{has\_index} & 1\\ \textsc{has\_name} & A\end{bmatrix}\right\rangle \\ \textsc{has\_tuple} & \begin{bmatrix}\textit{Tuple} & \\ \textsc{has\_arity} & 2 \\ \textsc{has\_tupleElement} & \left\langle\begin{bmatrix}\textit{TupleElement} & \\ \textsc{has\_index} & 2\\ \textsc{has\_value} & \begin{bmatrix}\textit{MurderWeapon} & \\ \textsc{has\_name} & \textit{knife}\end{bmatrix}\end{bmatrix}\right\rangle\end{bmatrix}\end{bmatrix}\\
\textsc{has\_negation} & \textit{false}
\end{bmatrix}$$

Figure 4.14: A sample *Condition*: *“A holds the knife”*

Relations make it possible to express logical assertions, and also conditions that may or may not hold over the ontological knowledge base. For this, a *Condition* concept exists. It has a slot *has_restriction* specifying a *Relation* instance that must hold for the *Condition* to be fulfilled. By using underspecified relations, *parameterized conditions* can be employed that take parameters as specified in the *has_argument* slot, and that take up the place of a correspondingly named tuple element. Figure 4.14 shows an example from the *Clue* system that expresses a condition with a parameter A that holds if *“A holds the knife”*. Dependent on a parameter A (which must be an instance in the ontology), the condition is true only if there is a *Relation* instance in the ontology that, if the first element of the condition restriction is replaced by A, unifies with the restriction. Conditions can also be negated via the value of the *has_negation* slot. A set of *Conditions* is interpreted as a conjunction, i. e., it is evaluated as true if all conditions in it are true. More complex conditions involving disjunction are expressed in a *Prolog*-like manner. They must first be transformed into disjunctive normal form (DNF) (Fitting, 1990). Then, a set of *Conditions* can account for each clause of the DNF.[12] The dialogue acts, dialogue games and activities that are described in the next chapter use *Conditions* to determine whether a given act, game or activity can be applied as an operator in a given situation, and which state of affairs will hold after the application.

4.4.5 Affordances

Addressing the difficulties in designing knowledge bases for agents that would allow them to perform in changing environments, (Doyle, 2002) describes the concept of attaching task-specific knowledge to the objects of an environment themselves, comparable to putting up a sign at a slot machine saying “insert coin to play”. We call such a piece of information

[12]Also similar to *Prolog*, the order of clauses resp. condition sets is relevant. If there are several sets of conditions that each trigger an action, the action associated with the first set that is satisfied will be selected.

an *affordance*. The idea is that this way, agents can be enabled to determine action options by way of examining their environment. Objects that are important in a scenario can be clearly annotated with additional interaction possibilities or just general description, while entities that are just included for a single purpose, or are just decorative, can be modeled more sketchily.

An advantage of such an approach is that the core functionality of an agent can be designed in part independently from the environment it will be used in; it just has to be able to interpret the information from object affordances and integrate it into its knowledge base. Consequently, the environment can change—even dynamically at runtime, while the agent is executing—and offer new possibilities, without a requirement to reconfigure the agent. Even when the environment is very large, only a portion of the world knowledge needs to be accessible in any given situation. This portion consists of the part of the world currently perceived by the agent, and the agent's internal state. The use of affordances also makes it easier to design the environment in an incremental fashion. (Peters et al., 2003) contrast the approach of agents employing low-level rules and a learning model to using *"smart objects"* derived from (Kallmann and Thalmann, 1998) that contain information about gestures relating to an object, object behaviors and attributes, and the behaviors of agents interacting with the object. However, affordances must be recognizable and interpretable by a dialogue participant that is to make use of it. Not all participants have to be able to discern all affordances, especially the human user will typically not be presented with a scenario of objects that have lots of signs attached telling the possible uses of the objects. On the other hand, the user is able to make her own inferences from the way objects are presented *visually* in the environment, and possibly already knows what can be done to something that looks like a slot machine, an advantage the virtual characters do not have.

$$\begin{bmatrix} \textit{FootballChampionship} & \\ \textsc{has\_affordance} & \begin{bmatrix} \textit{ExplanationProcess} & \\ \textsc{has\_explanation} & \begin{bmatrix} \textit{Explanation} & \\ \textsc{has\_cannedText} & \begin{array}{l}\textit{Die Fussball-Weltmeister-}\\ \textit{schaft findet dieses Jahr}\\ \textit{in Deutschland statt}\end{array} \end{bmatrix} \\ \textsc{has\_dependsOn} & \left\langle \begin{array}{l}[\textit{Explanation}],\\ [\textit{Explanation}]\end{array} \right\rangle \end{bmatrix} \\ \ldots & \end{bmatrix}$$

Figure 4.15: An object with an affordance slot in *VirtualHuman*

Since the action of our agents are in terms of acts, games, and activities, the affordances provide information about which acts, games and processes are possible with objects. An affordance can also specify which preconditions and postconditions are associated with it, e. g., a stone can be annotated with an action of "lift", but with the precondition that the action can only be executed if the agent is strong enough to lift the weight of the stone (which can be part of the factual knowledge of the character about the stone, or *also* be included in the affordance).

Affordances do not have to be the exclusive means to provide action possibilities for characters; they can also be used as a supplement to extend other knowledge of the characters.

In the tutoring scenario of the *VirtualHuman* system, we use a variant of objects with affordances to store knowledge that can be used by specialized as well as general processes. The affordances are stored in a special slot called *has_affordance* that can occur multiple times on any object. Another example of a general process taking advantage of such an affordance is that objects can have an *ExplanationProcess* object parameterized by an *Explanation* (see Figure 4.15) that provides an *Explain* dialogue act along with *depends_on* roles that point to other objects representing themes on whose understanding the explanation depends. A help process can then use this information directly to provide explanations, and elaborate recursively, or if the user asks for it. This feature was also implemented in the *VirtualHuman* system (see Section 7.2.7).

4.5 Summary

This chapter outlined the foundation of how we represent the knowledge base of the dialogue system and the characters. It identified the basic types of knowledge needed, which are concerned with the structure of the domain, knowledge about the world, the task, and the task state, as well as knowledge representing the discourse rules and history. It then described the approach of an ontological representation for modeling that makes use of an upper ontology as a starting point, and how it is translated to XML structures. A section is dedicated to important concepts and operations on the objects of the ontology. We introduced unification, restricted unification and overlay and explained the role of underspecification as well as how matching is done in our system. The process of integrating information into a private knowledge base (grounding) was explained. The following section was about the role of relations in the ontology and conditions that are used to test whether logical conditions are satisfied. Finally, the possibility to annotate objects directly with affordance information about what they offer in action possibilities for a character was discussed.

Chapter 5

A Model for Generating Multi-Party Conversational Behavior

5.1 Introduction

This chapter describes the model for conversational behavior that is the foundation for the conversation framework we have implemented. It covers interactions comprising of utterances that can be either spoken dialogue acts, contributions in other modalities, or combinations of both. The conversations are multi-party dialogues, i. e., they can be involve two or more participants, each of whom can either be a virtual character, or a human user. The basic interaction patterns apply regardless of whether the roles in the interaction are taken up by humans, or by computational agents. The model is designed to be adaptable to multiple configurations of applications. The number of system agents and human users may all vary; they can also change dynamically during an interaction, e. g., when participants enter or leave the conversation.

The actions and interactions of the conversation participants are described on several conceptual levels, and the communication between the parts of the software representing the virtual characters, and external application modules that are available to them, is also integrated. Therefore, communication protocols as used for, e. g., database or service queries can also be captured by the model. The characters can also carry out physical actions (in the virtual environment). This way, the entirety of action and communication in the system is covered. We do not aim for a system that is able to handle completely unrestricted and casual interaction. Instead, the model should provide a solid foundation for communication involving the two types of applications put forward in Chapter 2: task-oriented systems and interactive narratives. The foundation should be broad enough to provide for the basic interaction types found across such systems. In addition, when implementing a particular system, it should be easy to extend the basic interactions incrementally to accommodate any additional requirements.

We first describe the situation the dialogue participants are in. The virtual characters are *situated* agents in that they have a presence in a virtual environment that has a connection to the real world via multimodal communication channels. The characters can perceive other virtual characters, virtual objects, and the user via the communication channels, as well as influence the environment.

In the sections that follow, we describe the different levels of our model. We start with the bottom level concerning the dialogue acts that act as the atomic unit of action whereby the characters can influence the world. There are types of these acts that can be instantiated to yield a concrete dialogue act. We then show how the dialogue act types can be used as basic building blocks to define dialogue games that cluster acts together for more complex units of interaction involving more than one agent. As a third level of the dialogue model, we introduce processes that the agents can execute in order to reach their goals. Processes use the elements of the lower levels of the model, games and acts, and can also use other subprocesses as helpers. Finally, we turn to the question of how an agent can go about choosing the appropriate elements for its goals, and how different agents can cooperate on the several levels of communication.

5.2 Motivation and Basic Structure of the Model

5.2.1 Building Blocks

The underlying motivation for the model is to enable the dialogue designer to capture the structure of the intended conversation in an intuitive and uniform way, and to produce a library of building blocks that can be re-used across dialogue applications. Using these building blocks, we describe how the dialogue agents produce the behavior needed to reach their goals, and how they understand the contributions of other participants.

For this, we try to fit the possible conversational interactions into a set of common interaction patterns. Certain types of interactions are so frequent that they will occur in almost any conversation (for example, regular question-answer exchanges). These are used to provide a "standard library" of patterns for general interaction. Other patterns are less frequent and possibly non-occurring in a given corpus, depending on the interaction's context (for example, there is very low tolerance for rhetorical questions in radio-transmitted emergency communications, where precision, conciseness and unambiguity of the interaction is crucial). If a specific application turns out to require additional building blocks that are not covered by the general cases, the basic set of patterns must offer the possibility to be extended accordingly.

5.2.2 Layers of Action

The concept starts from the notion that an act is the minimal context-changing unit. Therefore, we have to start by explaining what we mean by the context. As sketched in Section 4.2.3, we make a distinction between the dialogue context, which is shared, and the context of the individual characters, which is private to each character. Seen from the perspective of a character, a context change can be internal, i. e., located in its own "mental state", or otherwise external.

Performed acts either influence the environment (we call this a *physical act*, even though it occurs in a virtual reality), or the mental state of other conversation participants. In the latter case, the act is a *dialogue act* in accordance with (Bunt, 2005) (cf. Section 2.4.1) in an instance of *communication*. Additionally, the mental state of a character can also be changed by inferences drawn internally by the character. The bottom layer of the model is concerned with physical and dialogue acts.

In the cases where a context change requires cooperation—or at least participation—from others, there must be a way for the participants to coordinate joint actions. For example, if a participant A wants to obtain some information, e. g., the current time, from B, it is generally not a good strategy for A to simply wait in hope that B at some time will feel the urge to tell about it. It is more promising if A *asks* B about it and thereby communicates to B the desire to obtain the information. However, this is only the case because A can rely on generally accepted social conventions that prescribe that the act of asking imposes an obligation on B to either provide the information in an *answer* act, or at the very least to give some justification why he is not inclined to do so (especially in cases of innocuous information such as the time). These conventions give conversation participants a means to predict, or at least presume, what behavior they can reasonably expect of other participants, and to plan their actions accordingly. The middle layer of the model employs such conventions to realize joint actions by using a version of the dialogue games introduced in Section 2.5.1.1.

In a structured and purpose-driven interaction, particular actions, intentions and needs of a participant are usually derived from a higher level of composite activities he pursues in order to accomplish some goals. For example, the desire for information in the previous example could be due to a goal of A to catch a train at a given departure time. To accomplish this goal, it might be necessary to construct and pursue a complex plan for an *activity* involving possibly numerous related communicative and physical acts (to find out where the train station is, how and when to get there, etc.). The model's top level is concerned with such goal-based activities that make use of the games in the middle level, and through them, the acts on the lower level.

5.3 The Dialogue System

As we have seen in Section 2.5.3, there are several architectural paradigms for modular dialogue systems that make different decisions regarding the interaction between modules and the division of labor. We assume a basic setup similar to the one introduced in the dialogue back-bone of the *SmartKom* system. This means that the system is indeed heavily modular, there is a possibility for each module to send messages to any other module at any time (Chapter 6 will describe the actual mechanism used in the framework).

The number of other modules that need to interoperate with the conversation manager may vary considerably, depending on the system architecture. In the *SmartKom* system, there are eight modules that directly communicate with the conversation manager (see Figure 2.29), while in *OMDIP*, there are only three. Figure 5.1 shows the main information flow in the system, starting from the recognition of user utterances to the realization of system presentations by the player or other effectors. Both the multimodal input and output pass through the *Discourse Modeler* module that is located between the analysis and generation modules. The role of the discourse modeler is explained in Section 5.3.2. The conversation manager also communicates with modules that are frontends for the applications provided by the system (if any), and possibly other system modules as needed.

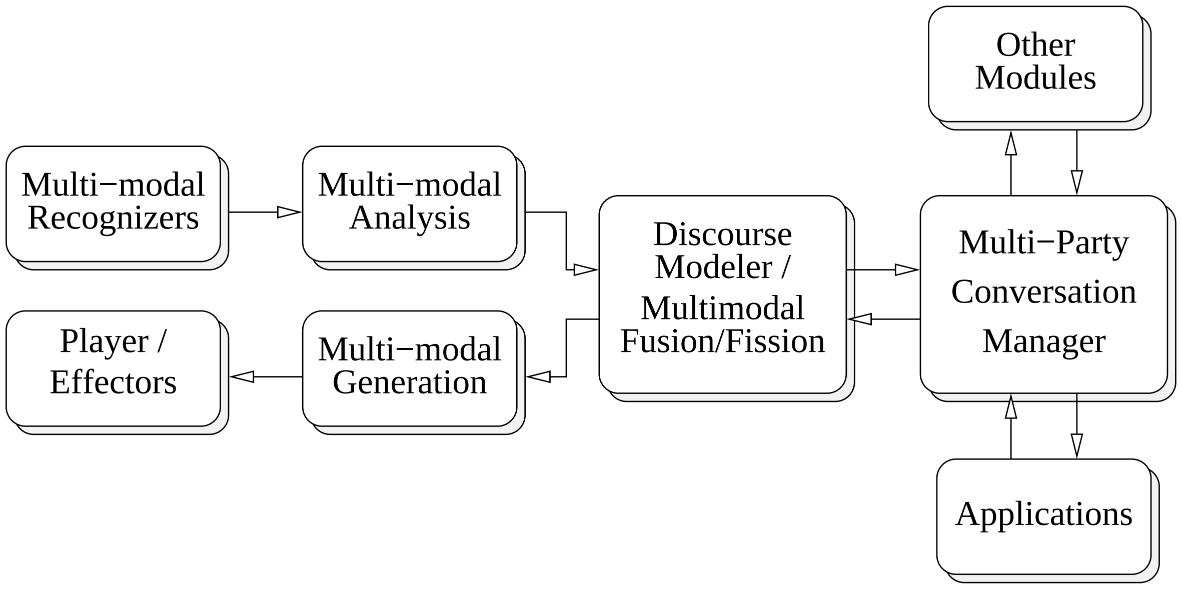


Figure 5.1: Main information flow assumed for the dialogue system

5.3.1 The Multi-Party Conversation Manager

Regarding the interaction, we make a distinction between a so-called *objective environment* representing the actual state of affairs, and a number of *subjective environments* that represent the individual views of the participants in the multi-party situation. The environments are represented by different ontologies representing the entities that are present in the scenario, and relations between them.

The conversation proceeds in *turns*, each consisting of one or more multimodal utterances. The entity instances in the ontologies are dynamic and subject to change. Instances may be added, removed, or modified as a consequence of turns in the conversation or other events. The ontology used to represent the objective environment is called the *system ontology*. Each character under the control of the conversation manager has a *character ontology*. The representations of the environment and the participants are maintained by the conversation manager and kept clearly separated. A *controller* component in the conversation manager dynamically creates and administers an individual dialogue engine for each participant, called its *Conversational Dialogue Engine*, or *CDE*. The CDEs interact with the rest of the system by sending and receiving messages that contain communicative acts.

The controller is responsible for distributing these messages to the designated recipients. Messages that enter or leave the conversation manager are inputs gathered by multimodal recognizers, outputs delivered to multimodal rendering engines, and communication exchanges with other modules. Those external interactions go via so-called *communication channels*. Inbound and outbound channels are called *input* and *output channels*, respectively. The channel device is used as an abstraction to encapsulate that external communication may use different information representations (i. e., other than communicative acts), and in this case, it is necessary to apply conversions to and from the uniform representation used within the conversation manager when messages enter or leave it.

We define

Definition (Multi-Party Conversation Manager)
A *Multi-Party Conversation Manager* M is a software artifact parameterized by

- Ω_M, an ontology called the *system ontology*,
- $\mathcal{C}_M$, a set of *Conversational Dialogue Engines* (Section 5.4.1),
- $\mathcal{I}_M$, a set of *input channels* (Section 5.3.3), and
- $\mathcal{O}_M$, a set of *output channels* (Section 5.3.3).

Apart from the controller, each component of the conversation manager is dependent on the particular dialogue system.

5.3.2 The Discourse Modeler

Similar to the approach in *SmartKom*, we separate the conversation management task from modeling the discourse structure and the discourse history. The division of knowledge involved roughly corresponds to the distinction between PUBLIC and PRIVATE parts in information state approaches like *TRINDI* (see Section 2.4.3.3). In our approach, the processing of both kinds of knowledge is assigned to two specialized modules. This also means that the conversation manager relies on a component that preprocesses the multimodal user input. It is assumed that this component does multimodal fusion and reference resolution, and integrates it into a model of the discourse history. We use the *FADE* (Fusion and Discourse Engine) toolkit by Pfleger (Pfleger, 2007) for this task.

Figure 5.2 shows the functional architecture of *FADE* including its two main subcomponents, *PATE* and *DiM*. The left part of the figure shows the conversational context and the right part shows the discourse context and its API for processing propositional contributions. *PATE* uses a production rules system to process interactional events such as eye gazing and the start and end of utterances, and is responsible for generating reactive behavior, such as, e. g., gazing in response, other backchannel feedback, and generating gestures while waiting for a turn. It also does initial addressee identification. *DiM* maintains the discourse history and applies modality fusion, reference resolution, and utterance enrichment from the context to transform the input events (i. e., the semantic content of the utterances as delivered by the analysis modules) into dialogue acts. Messages containing these dialogue acts are sent as input to the conversation manager. *FADE* also postprocesses the output of the conversation manager and enriches it with additional reactive behavior. Inputs and outputs are both integrated into the discourse history.

FADE uses the same internal knowledge representation as the conversation manager, i. e., typed feature structures that represent ontological instances with a taxonomical ordering which is shared between the modules. The exchange of messages with *FADE* is therefore very efficient, since both modules are tightly coupled and there is no need for message conversion. *FADE*'s preprocessing and postprocessing makes it possible for the conversation manager to be modality-agnostic on the input side; however, when producing output, the conversation manager still has the opportunity to select specific modalities. We will not describe the operation of *FADE* in greater detail here, but refer to the comprehensive treatment in (Pfleger, 2007).

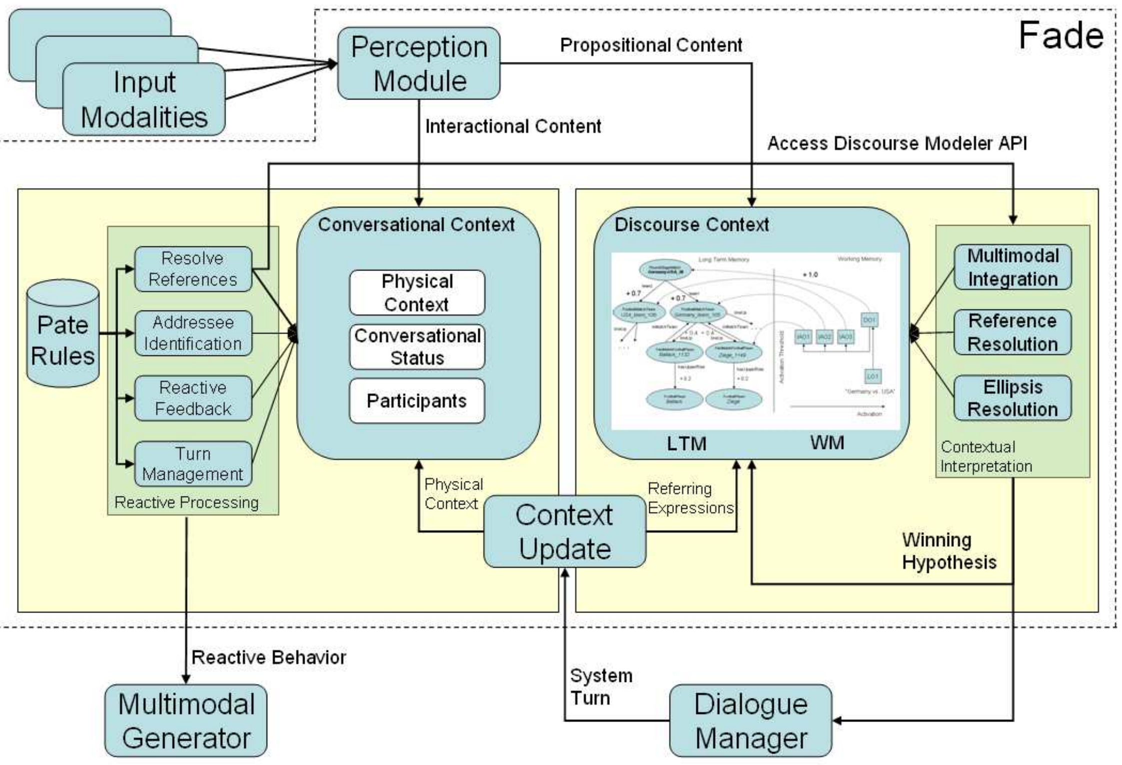


Figure 5.2: The functional architecture of *FADE* (Pfleger, 2007, p. 186).

5.3.3 Communication With Other Modules: Channels

Apart from *FADE*, other modules in the dialogue system generally do not use the XML-Ontology representation for communication. This is especially the case when off-the-shelf modules (e. g., a commercial TTS engine) or already existing applications with a fixed interface protocol are used. To be applicable in system setups with differing message formats, the model needs an abstraction to transform input from and output to such modules into terms of the XML-Ontology. This is done by routing input and output via *channels*. Channels are functions doing a translation, or, in their implementation as software artifacts in the framework, transformation filters through which the messages are passed. There are input channels for translating incoming messages to the internal format, and output channels doing the inverse translation for outgoing messages. An input channel can be subscribed by any number of CDEs which will receive translated messages from other modules, and each CDE can publish messages for an output channel, which will be delivered in translated form to one or more receiving module.

Definition (Input Channel)
An *input channel* $I \in \mathcal{I}_M$ for a conversation manager M is a software artifact parameterized by

- INCOMING, a set of sources for incoming data connected to I,

- SUBSCRIBERS $\subseteq \mathcal{C}_M$ is a set of CDEs of M that subscribe to messages via I,
- *input*, the *input translation function*, is a function

  $$\textit{input} : \textsc{Strings} \rightarrow i^*(\Omega_M)$$

  that maps string input from all data sources $i \in$ INCOMING to instances in the ontological representations in Ω_M.

Definition (Output Channel)
An *output channel* $O \in \mathcal{O}_M$ for a conversation manager M is a software artifact parameterized by

- OUTGOING, a set of sinks for outgoing data connected to O,
- PUBLISHERS $\subseteq \mathcal{C}_M$, a set of CDEs of M that publish messages via O,
- *output*, the *output translation function*, is a function

  $$\textit{output} : i^*(\Omega_M) \rightarrow \textsc{Strings}$$

  that maps instances in the ontological representations in Ω_M to string output for all devices $o \in$ OUTGOING.

In some setups, it can also be necessary to let input and output channels do more complicated transformation work, especially with regard to application protocols. For example, in *SmartKom*, one input channel connected to the function modeler had to combine two asynchronous streams (one containing operation success status, the other data content from the application) in order to construct one feedback message to the conversation manager.

5.4 The Conversation Participants

The model covers three kinds of conversation participants: Virtual characters, human users, and other modules of the system which can act as “invisible” participants (i. e., they are not represented by a graphical avatar, and their communicative acts are not rendered by the presentation). An example for the last kind is the narration engine in *VirtualHuman* that operates like a director in a dynamic play and triggers and coordinates story goals. The participants are treated uniformly by the conversation manager’s controller: Each is assigned a dedicated dialogue engine that represents it in the model, and handles the conversation acts that affect the participant.

5.4.1 Conversational Dialogue Engines

A CDE uses an ontology to represent the participant’s view of the domain. Its set of instances may be different from the system ontology. However, all ontologies must be compatible in the sense described in Section 4.3.4, to ensure that instances from one of them can be interpreted in the taxonomical structure of all others. The participant itself is represented by an entity of the concept *Agent* from the system ontology, which has the subconcepts *Character* (for virtual characters), *User* and *MetaCharacter* (used for, e. g., a director module). These instances can hold additional information about the character and model, e. g., personal attributes such as age, sex, and profession, and character traits that may be used to parameterize its behavior. Characters can also maintain their own models of other participants using *Agent* instances.

A CDE that implements a virtual character has to produce the behavior for this character, and process the events that the character can perceive. A human user's CDE has the task of relaying her utterances that are entering the conversation manager via input channels connected to the analysis modules, and sending them to other CDEs in the same fashion as if the utterances were produced internally in the CDE. A user's CDE can generally ignore utterances from other CDEs that are addressed *to* it, since these will be realized by the presentation module and can in this way be perceived by the user.[1] CDEs representing other modules are similar to user CDEs in that they relay messages from and to their modules.

The actions of character and user CDEs manifest in the environment in an identical way, except that again, user utterances, unlike character utterances, are not sent on to the player component to be rendered, because otherwise the system would echo user utterances. A fundamental difference, however, is that the human users—which are not part of the computational system—of course produce their own behavior, whereas CDEs for virtual characters must also provide some reasoning mechanisms to generate their actions. To describe the actions that a participant is capable of, its CDE also comprises sets of activity types, dialogue game types, and act types it can perform. We define

Definition (Conversational Dialogue Engine (CDE))
A *Conversational Dialogue Engine* $C \in \mathcal{C}_M$ in a conversation manager M is a software artifact parameterized by

- Ω_C, an ontology called the *CDE ontology*. Ω_C must be compatible with the ontology Ω_M of M,
- $p \in i_{\mathit{Agent}}(\Omega_M)$ is the *participant instance*,
- $\textsc{ActivityTypes} \subseteq i_{\mathit{Activity}}(\Omega_C)$ is a set of *activity types* (Section 5.6.3),
- $\textsc{GameTypes} \subseteq i_{\mathit{DialogueGame}}(\Omega_C)$ is a set of *dialogue game types* (Section 5.6.2),
- $\textsc{ActTypes} \subseteq i_{\mathit{Act}}(\Omega_C)$ is a set of *act types* (Section 5.6.1),
- *registeredServices* is a function

  $$\mathit{registeredServices} : \textsc{ActTypes} \longrightarrow \textsc{ActivityTypes}$$

  that maps from act types to activity types.
- f_C is a *perception filter* for Ω_C (Section 5.4.2)

The components of a CDE include knowledge sources (comprising the ontology, the types of activities, games, and acts, as well as the ontological instance representing the character

[1] In some cases, user CDEs have to process other (non-communicative) messages. In *VirtualHuman*, the director CDE sends a message to one user CDEs to disable the input microphone during the second quiz phase.

itself). The other elements are required for the operation of the engine: The registered service mapping is used to find the appropriate activity type for perceived utterances, and the perception filter influences how the character perceives utterances and other events in the environment. The following sections explain the function of the elements of the CDE definition.

5.4.2 Perception Filters and Grounding

The dialogue engines of the participants contribute to the interaction exclusively by triggering and perceiving actions via the virtual environment. The actions are represented by instances of the ontological class *Act*. Whether and how an act is perceived by a conversation participant can depend on the circumstances.

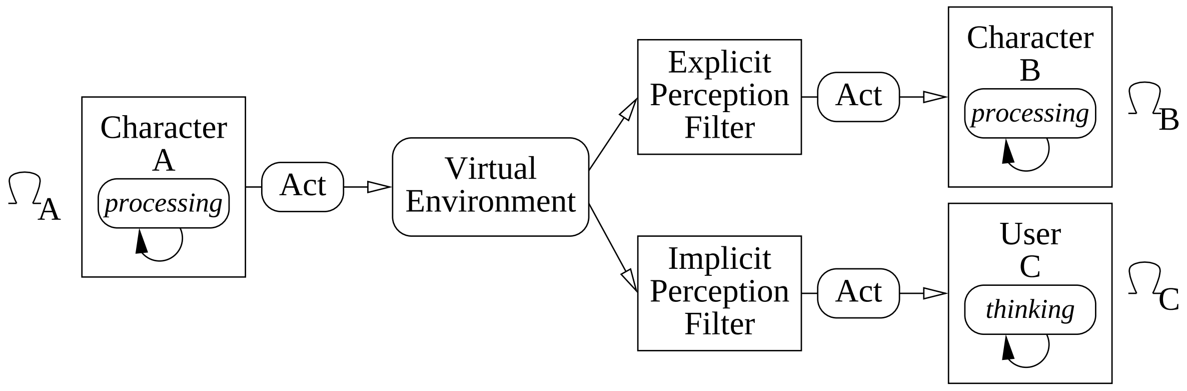


Figure 5.3: Conversation participants exchanging acts

Figure 5.3 illustrates the way acts are passed from an initiating character to other conversation participants. Even though the characters are situated in a virtual reality, they are not able to perceive everything that happens in their environment. For one, the mental states of the other virtual characters (or human participants, for that matter) are encapsulated and cannot be directly accessed. Also, the "physical" events in their surroundings are also not necessarily open to them. In an environment consisting of several rooms, such as the *Clue* scenario, actions happening in a separate room cannot be seen, spoken utterances that happen too far away might be misunderstood or not heard at all, etc. Another possibility is to account for whether a participant is paying attention, or whether he is possibly distracted, e. g., by another strand of conversation.

There needs to be a mechanism that restricts what can be perceived by the participants. We call such a mechanism the *perception filter* for the participant. A perception filter in our model acts as a function that maps an act happening in the environment to perceived acts for a CDE. It employs a set of conditions to determine whether a fact or event in the environment can be perceived by the character (for example, whether the distance from the hearer to the producer of an utterance is below a set threshold, or whether there are no obstacles such as virtual walls in between).

The filter may remove or distort information. For the human users, the presentation components act as *implicit perception filters* when, e. g., speech synthesis renders conversation contributions by spatially distant characters with a lower volume, or the graphical presentation does not render events that are not be visible from the point of view of the user. The corresponding mechanism for a virtual character applies *explicit perception filtering* using rules that determine how acts should be perceived by the character, or may altogether prevent them from arriving by mapping them to an empty act.

We define a perception filter as follows:

> **Definition (Perception Filter)**
> A *perception filter* for an ontology Ω is a function
>
> $$f : i^*_{Act}(\Omega) \mapsto i^*_{Act}(\Omega)$$
>
> that maps act instances from Ω to perceived act instances in Ω.

When acts are perceived by other conversation participants, they can have an effect on their mental state (beliefs, desires, intentions and obligations). This means, if a participant perceives an act, it will *process* it (in the case of a human participant, we would rather say she *thinks* about it), and will possibly react by producing her own acts. There are two perspectives to this: what the initiator of an act intends for it to accomplish, and what it actually does change in the contexts of the addressees (or possibly additional overhearers). For example, by making a statement of some fact, the initiator usually intends the effect that the addressees include the fact in their set of beliefs. However, it may be that the intention is not successful for some reason, for example

- addressees do not trust the initiator enough and reject the statement,
- addressees do not know how to integrate the statement into their belief bases, or
- the act does not correctly arrive at an addressee because its content is wholly or partially eliminated by a perception filter.

Conventionally, the participants need to agree on some *common ground* about what has been established between them during the interaction. The common ground states public acceptance: it can be different from the actual beliefs of the participants. With autonomous characters, it is also possible that a character mistakenly assumes some fact to be part of the common ground when it actually is not. To raise its confidence that its assumptions are correct, it may request explicit feedback by the other characters. Whether or not a virtual character decides to accept the semantic content of an utterance is determined by the rules of the dialogue game it occurs in; this will be explained in later sections of this chapter. If it does, it has to integrate the content into its own ontology, i. e., ground the utterance for itself. This grounding operation was described in Section 4.4.3.

5.5 Interaction

In Section 2.4.1 (page 35), we cited Austin's definition of dialogue acts as context-changing actions. What an interaction is about does not stem from the single actions alone, but also

Ontology states	
System ontology state	“objective” context for the system, state of the environment
Character’s ontology state	context for one character, private view of the state of the world
Building blocks	
Act	elementary context change
Dialogue Act	context change for other DPs: communication
Dialogue Game	context change in cooperation with other DPs using shared rules that allow to anticipate what other DPs will do
Activity	complex goal-directed context change, possibly involving multiple DPs in different roles

Figure 5.4: Summary of the relations between the roles of the different ontology states and building blocks

needs to take into account the relationships between them, and the reasons why they are done.

In Section 2.4.2, we identified the concept of a DSP and subsequently dialogue games as a means to connect related dialogue acts that serve as a sub-unit for a DSP. In turn, dialogue segments are undertaken for some superordinate purpose, such as realizing an intention. Figure 5.4 summarizes these notions again.

5.5.1 Layers of Conversational Action

We organize the actions of the conversation participants hierarchically in three different layers of abstraction. The layers are those of *activities*, *dialogue games*, and *dialogue acts*. An action of a given layer can trigger and coordinate actions located on the same layer or on lower layers, but not above its own layer. We use the example in Figure 5.5 as a reference for the following explanation.

The layer structure is similar to the levels of intentional structure given in (Alexandersson, 2003). It is, however, different in several respects. First, it does not include a separate level for the (single) toplevel activity (the dialogue level); rather, the toplevel activity is treated like any other activity. Second, it does not have different levels for dialogue moves and dialogue acts. The introduction of the move layer was motivated by the observation that *“frequent occurrences of sequences of utterances [...] can be viewed as a unit”* (Alexandersson, 2003, p.57). In our model, dialogue moves comprising more than one dialogue act correspond to portions in a dialogue game where the initiative remains with the same participant over more than one act; we do not think that these cases necessitate the introduction of an additional modeling level. The *surface realization* form also does not constitute a separate level in our model, since the actual generation and multimodal fission of the acts is not part of the conversation manager’s operation.

Consider as an example the following fragment of a sales dialogue between a bakery shop assistant and a customer:

(1)	SHOP ASSISTANT:	*Good morning!*
(2)	CUSTOMER:	*Good morning.*
(3)	SHOP ASSISTANT:	*What would you like?*
(4)	CUSTOMER:	*That depends. Do you have any blueberry muffins today?*
(5)	SHOP ASSISTANT:	*No, but we have chocolate muffins [↗] and plain muffins [↗] over here.*
(6)	CUSTOMER:	Then I'd like [↗] three chocolate muffins and [↗] a bottle of milk.

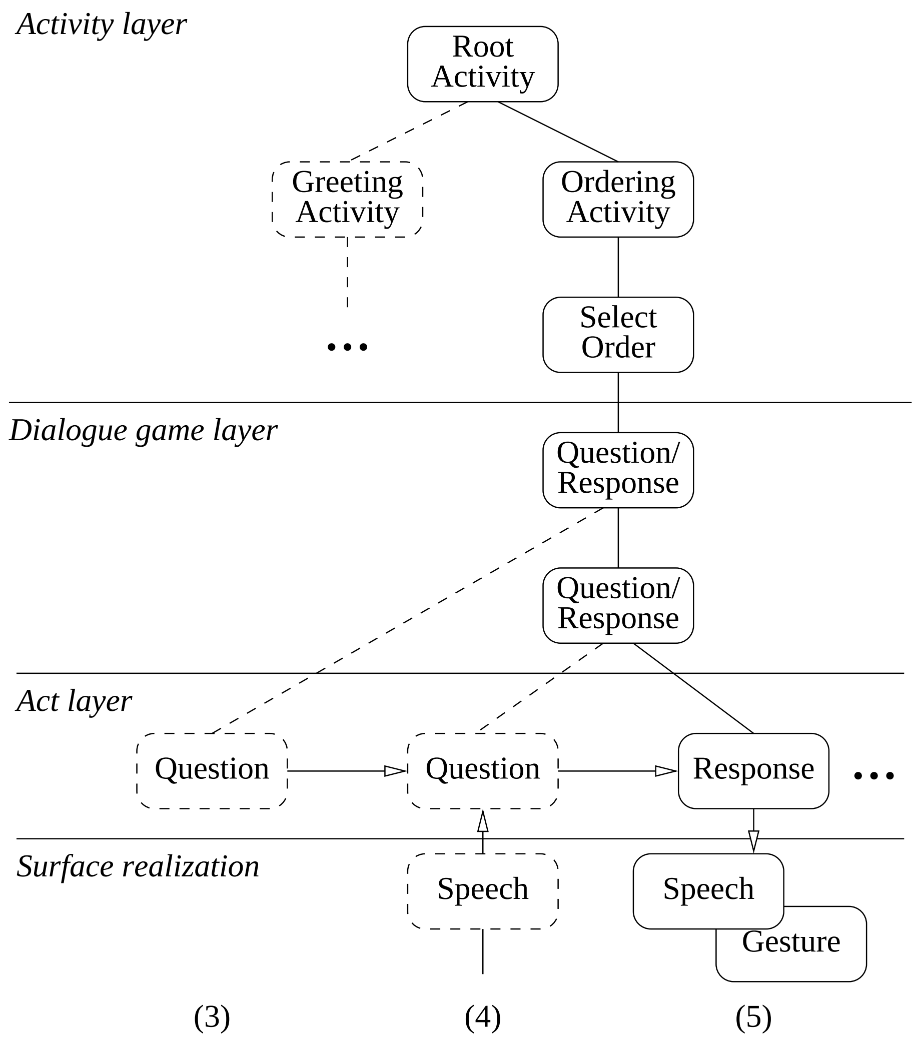


Figure 5.5: Example process hierarchy in the interaction in a sales dialogue

Figure 5.5 shows how the process hierarchy could look like at some point for a character that implements a corresponding shopping application. In the example, the interaction starts with an activity where the participants exchange greetings, and then the shop assistant proceeds to ask what the user is interested in. Instead of answering right away, the user also has the option of asking a counter-question. The figure shows a situation where a character is responding to such a counter-question (the items shown in boxes with dotted lines have already been completed at this point). The layers shown have the following functions:

- **Activity Layer**

  On the highest level, the activities represent the (individual) courses of action of a participant. Executing an activity can involve starting sub-activities. In the example, the *Greeting* activity (shown as a dotted box) has already been completed and the *Ordering* activity is currently being pursued. This activity will consist of the task of selecting an order along with several parts not shown here, such as specifying a delivery method, and confirming the order. Each of these parts is a sub-activity to the original activity. The *Ordering* activity triggers them to delegate part of its functionality and may block while they are executing. In the figure, the *Ordering* activity has triggered the sub-activity *Select Order* to gather the information about what item the user wants to order.

  The *Root Activity* shown topmost in the figure is the parent activity for all other activities and spawns them as needed. As will be explained later, the root activity is also used as a fallback to deal with situations that no other process is prepared to handle. Activities implement the behaviors of the characters directed to achieve the different goals of the characters.

- **Dialogue Game Layer**

  To engage in interactions with other conversation participants or other modules of the dialogue system, activities start and join into dialogue games that are located in the middle layer. Executing a Dialogue game may also involve starting subgames. In the example, a *Question/Response* game has been started by the active *Select Order* activity. After the question was posed, a counter-question was asked by the addressee of the question, initiating a nested *Question/Response* subgame (with switched initiator / responder roles). As with activities, dialogue games also usually block while triggered subgames are still active. Dialogue games realize the rule-governed exchange of dialogue acts.

- **Act Layer**

  The actual dialogue acts reside on the bottom layer. They are the atomic units of communication between the interlocutors, and the physical acts the characters can perform. When a dialogue game is active, it generates dialogue acts that are sent to their respective addressees. On the surface level of multimodal presentations, it is possible that one dialogue act is rendered as several distinct actions. This will, e. g., be the case if a multimodal fission component decides that the content is to be rendered as a combination of speech and gestures. The surface realizations may overlap, but unlike activities and dialogue games, dialogue acts cannot be active in parallel, but are realized one at a time by each character. Section 7.2.4 talks about the treatment of overlapping surface realizations in the *VirtualHuman* system.

  We focus on (multimodal) utterances in conversations, and the model is primarily intended to describe interaction by communicative acts. However, physical acts such as moving an object and subsequently receiving perceptions about the effect can also be seen (metaphorically) as a “dialogue with the world” and we treat them as such in our system: If a character produces a physical act, e. g., it tries to open a door, it receives back an act containing a perception information that the door is now open—or not, if the action failed (see Section 5.6.1.2 on physical acts).

We also call elements of the two upper layers *executable acts* because they are of procedural nature in that they may take some time to complete, and may be interrupted, suspended, and continued. On the other hand, actions on the lowest layer are performed as atomic actions, i. e., without the possibility of being interrupted, although they do also take some time to complete and/or may be realized as more than one surface act.

5.5.2 Motivation for Action

What exactly incites a character to perform an action? Consider the examples in Figure 5.6. The list proceeds from automatic, subconscious actions over actions that are governed by reflexes to behavior that involves explicit deliberation. At the most basic level, functions such as breathing are performed automatically, without consciousness even taking notice in the general case. The wincing in example (2) is a classic reflexive reaction, which will be registered by consciousness after the fact only. Gazing while talking, as well as gazing back when talked to as in (3) also occurs automatically; it is a deeply ingrained social behavior that requires no explicit decision to be performed. Case (4) can occur as a deliberate action or as a reflexive one (especially in the case of the person that gets offered the handshake). The examples (5) and (6) are deliberate actions. In the latter case, the action is undertaken as part of a superordinate plan to accomplish something, i. e., it is part of some overall deliberation going beyond the action itself.

automatic	(1)	Peter is breathing.
	(2)	Peter winces at the sound of the explosion.
reflexive	(3)	Peter looks at Mary while addressing her.
	(4)	Peter and Mary shake hands.
	(5)	Peter answers Mary's question.
deliberative	(6)	Peter calls Mary so they can lift the heavy box together.

Figure 5.6: Different types of action motivations

We are mainly concerned with action that is the result of goal-driven deliberation. However, our framework also provides support for automatic and reflexive action, such as the characters' blinking behavior, as was done in *VirtualHuman*.

Different reasons can lead to the adoption of a goal by a virtual character. Based on the source and the purpose, we distinguish between *external goals*, *internal goals* and goals that arise from the interaction between the conversation participants, which we will call *interaction goals*.

- **External Goals and Narrative Mode**

  In the task-oriented situation, the (cooperative) conversation participants share the overall joint goal of accomplishing tasks together. There may be a large number of possible tasks, and it is possible that it is not known at the beginning which tasks will be needed. Usually the system's purpose is to provide help for the human user, and users choose the goals for the interaction, whereas the virtual characters will only introduce new goals that help to advance the goals of the user. If the system only has one purpose, or determining the desired task is a task itself, the system can also take the initiative in setting goals.

In interactive narratives, the overall goal is to arrive at the story conclusion via a set of story points. If a narration engine is controlling the storytelling, it exerts the control over the goals of the virtual characters. The characters do not necessarily cooperate with the human users. Depending on their role in the story—e. g., being the antagonist—they might even work to jeopardize the human user's goals.

If the main goals are set by an external narration engine, the system is running in *Narrative Mode*. In this mode, the character's behavior is only semi-autonomous: An external goal commits the character to performing an activity. Additional parameters can be given by supplying values for roles of the activity or constraints on its execution, e. g., a time limit.

- **Internal Goals**

  A goal can also arise from the internal state of a character, e. g., from its desires or as a consequence of other goals. In some scenarios, such as the military setting of the *MRE* system, it is also possible that a character has a social status that allows it to give an order to other characters, which will then adopt a goal to follow the order based on its internal obligations.

- **Interaction Goals**

  Finally, a goal can be adopted as a reaction to a communicative action of another conversation participant, for example a question. It can entail the execution of a new activity, or cooperatively pursuing a joint action (such as resolving the question).

During an interaction, when a conversation participant adopts a goal, it will engage in an *activity* whose purpose is to fulfill the goal. We base our definition on Allwood's characterization of social activities as referred to in Section 2.4.2.1 and also require procedural information to "define what the activity is all about" (Allwood, 2000). The model describes the context for the environment and for the individual characters. The interaction is a sequence of actions, mostly communicative ones, that change these contexts. In turn, context states give rise to desires and obligations to perform further acts. If there were no external influence, such as input from users and the narration engine, and no dynamic desires, it would be possible that the system would at some point reach a stable state (barring obligation loops).

There are two kinds of basic obligations for a character. First, a character is obliged to try to continue dialogue games it is involved in. Second, a character must take action to fulfill its goals. If both kinds of obligations conflict, there must be some mechanism to resolve the conflict, e. g., by assigning priorities determining which obligations are to be addressed first. A special case is when a goal is a subgoal of another: If an activity or game g is dominated by another activity or game g', then its execution has priority over the execution of g'.[2]

[2]The Gricean Maxims for cooperative conversation (Grice, 1975) do not apply on this level. It is concerned

5.5.3 Action Modes

The two upper layers of the action model—activities and dialogue games—are of a composite and procedural nature, while acts and their surface realizations do not have an internal structure. In contrast to atomic actions, executable acts remain active for some time. There are the two categories of action: observable ”behavioral” action, and non-observable deliberative or regulatory mechanisms. The former category manifests as communication and physical acts, the latter constitutes internal computations of the agent.

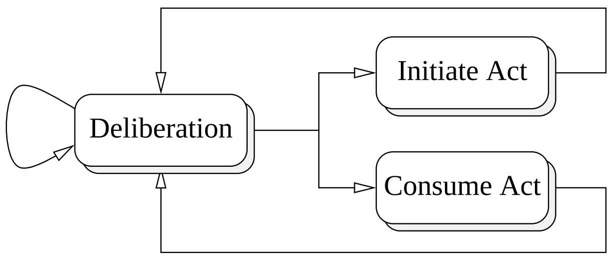


Figure 5.7: Action cycle

This gives us three modes of activity: Deliberation, passively consuming acts, and actively initiating acts. In an interaction, the three modes occur in an interleaved fashion. The current mode of a character can change as depicted in Figure 5.7. The circumstances in each mode of activity are the following:

- **Deliberation**

  In this mode, the character examines its knowledge state and performs computations in order to determine its future action. This can include planning and adopting subgoals, knowledge base manipulations, and updates of the character model. If no intention is formed to go into initiative mode, and no communication is perceived from other characters, the character remains in deliberation mode. Also, the success conditions of the currently active executables are checked to determine whether they can terminate.

- **Initiate Act**

  The character initiates communication or action in order to progress towards reaching its active goals, i. e., it acts on its intentions, and then goes back into deliberation mode. The actual production of an act may require additional preparatory action, e. g., to acquire the floor before speaking, or waiting until the realization is finished (cf. Section 7.2.4).

- **Consume Act**

  The character perceives communication acts from other dialogue participants, changes in the environment, or messages from external modules. It has to integrate the content into its knowledge base, and the acts have to be delegated, in the context of the active

only with “well-formedness” of dialogue games and does not consider the *content* of the utterances. Furthermore, characters do not have to be cooperative (especially in narrative settings): it could be explicitly desired that a character’s utterances should be, e. g., untruthful, ambiguous, or irrelevant.

goals, to a suitable executable that is able to handle it. External messages (e. g., from a narration engine) can also lead directly to the adoption of new goals.

This concludes our initial overview of the operation of the conversation manager in the model. Before elaborating on it in Section 5.7, we describe the building blocks that constitute the different layers of action.

5.6 The Building Blocks of the Model

The ontology provides the building blocks used to assemble interactions in the three layers we introduced in Section 5.5.1. They represent *types* of actions and are called *act types*, *dialogue game types*, and *activity types*, corresponding to their associated layer. When they are used in an interaction, concrete *instances* of these types are created. For example, a character could be concerned with several instances of the same activity type, e. g., to answer different questions, in parallel or at different stages of the interaction. On the other hand, a concrete instance of a question answering activity starts and ends at given points in time and features concrete values for different roles, or parameters, including who is performing the activity, on whose request, and what precise pieces of information the act is about.

A building block type is an underspecified ontological instance that contains the general properties of the building block. For example, in a multi-party situation, the act type for a *QuestionIf* dialogue act would specify that it involves the roles of initiator of the question, one or more addressees, and some content that the question is about. It acts as a schema or template for uses of the *QuestionIf* concept. When used in the conversation, a fully instantiated *QuestionIf* building block has these roles filled. The same principle holds for activities and dialogue games. The building block types are instances of entities, and should not be confused with concept types on the ontological level (there can be an arbitrary number of building blocks of the same concept type). One property of all building blocks is that they can specify *preconditions* and *postconditions*, which are sets of *Condition* instances. This is important because an agent needs to know under which circumstances a building block can be used, and what effects are to be expected after it has been used.

The dialogue designer can, for example, define a an act type for a “rhetorical question” to be a building block as shown in Figure 5.8. This act type can then be used as a template for all rhetorical questions. It has parameter roles that specify how slots are filled in an instance of it. In the example, the subject ($\boxed{1}$) of the question is assigned to the slot *has_content* by the parameter *subject*. Two special preconditions are imposed that state that the knowledge base contains a relation *knowsIf* which holds for two tuples: one holding the values of the addressee and subject roles of the question, the other the values of the initiator and the subject roles. Conditions are always evaluated against the knowledge state of the character doing the evaluation. In other words, an initiator would find the precondition of a rhetorical question satisfied if it believes that it knows the answer, and believes that the addressee knows it, too.[3]

[3]The given definition does really capture just one kind of rhetorical question. For example, utterances of the form *“how long do I have to tell you to do X?”* or *“would you open the door for me?”* are also considered rhetorical questions.

$$
\begin{bmatrix}
\textit{QuestionIf} & \\
\textsc{has\_initiator} & \textit{Character} \\
\textsc{has\_addressee} & \textit{Character} \\
\textsc{has\_precondition} & \left\langle \begin{matrix}
\begin{bmatrix}
\textit{Condition} & \\
\textsc{has\_argument} & \langle \boxed{1}, \boxed{2} \rangle \\
\textsc{has\_restriction} & \begin{bmatrix}
\textit{Relation} & \\
\textsc{has\_name} & \textit{knowIf} \\
\textsc{has\_tuple} & \begin{bmatrix}
\textit{Tuple} & \\
\textsc{has\_tupleElement} & \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{0} \\ \textsc{has\_value} & \boxed{1} \end{bmatrix} \\
\textsc{has\_tupleElement} & \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{1} \\ \textsc{has\_value} & \boxed{2} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}, \\
\begin{bmatrix}
\textit{Condition} & \\
\textsc{has\_argument} & \langle \boxed{1}, \boxed{3} \rangle \\
\textsc{has\_restriction} & \begin{bmatrix}
\textit{Relation} & \\
\textsc{has\_name} & \textit{knowIf} \\
\textsc{has\_tuple} & \begin{bmatrix}
\textit{Tuple} & \\
\textsc{has\_tupleElement} & \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{0} \\ \textsc{has\_value} & \boxed{1} \end{bmatrix} \\
\textsc{has\_tupleElement} & \begin{bmatrix} \textit{TupleElement} & \\ \textsc{has\_index} & \textit{1} \\ \textsc{has\_value} & \boxed{3} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{matrix} \right\rangle \\
\textsc{has\_parameter} & \left\langle \begin{matrix}
\boxed{1} \begin{bmatrix} \textit{FormalParameter} & \\ \textsc{has\_name} & \textit{subject} \\ \textsc{has\_slotName} & \textit{has\_content} \end{bmatrix}, \boxed{2} \begin{bmatrix} \textit{FormalParameter} & \\ \textsc{has\_name} & \textit{addressee} \\ \textsc{has\_slotName} & \textit{has\_addressee} \end{bmatrix}, \\
\boxed{3} \begin{bmatrix} \textit{FormalParameter} & \\ \textsc{has\_name} & \textit{initiator} \\ \textsc{has\_slotName} & \textit{has\_initiator} \end{bmatrix}
\end{matrix} \right\rangle \\
\textsc{has\_content} & \textit{:THING}
\end{bmatrix}
$$

Figure 5.8: An example act type representing a rhetorical question

The building block still lacks information about, e. g., the actual content, or the addressee, of the question. To create a concrete "rhetorical question" instance, this information needs to be added. The same holds for the other two kinds of building blocks. As ontological objects, the building blocks and concrete instances made from them are used in a *representational* way to hold information about the event or event type they stand for. The building blocks of the two upper levels, activity and dialogue game types, also have a *procedural* aspect. While elementary acts can realize behavioral action, the upper levels are purely deliberative.

We now show how the different building block types are represented, which should also already give an idea of what they stand for. The subsequent sections then explain the procedural view, i. e., how they are used by the CDEs to conduct an interaction. This is illustrated by an example taken from a session with the *VirtualHuman* system.

5.6.1 Act Types

Acts are communicative and physical events that are atomic from a CDE's point of view. This does not mean that they occur instantaneously, or that they correspond to a singular event in the virtual world. The realization of spoken utterances and gestures takes some time, and a CDE implementation must account for this (see Section 7.2.4). Also, a single act will often be realized as a combination of world events, especially in a multimodal system, e. g., if an utterance is accompanied by a pointing gesture or facial expression. However, the CDEs cannot produce units smaller than an act, and in turn, they receive multimodal utterances in the form of single acts that are the output of multimodal fusion.

The base ontology specifies several subtypes of acts; a subset of their inheritance tree originating from the *Act* concept is shown in Figure 5.9. For interactions in the form of a multi-party conversation, the main type of act is a *DialogueAct*, which is a subtype of *CommunicativeAct*. The *DialogueAct* type has a number of subtypes that correspond roughly to categories from different dialogue act tag sets from linguistic theory (e. g., (Carletta et al., 1996); see Section 2.4.1). It is not the goal to be complete or sound from a speech-theoretical point of view, but rather as an extensible starting point to collect the act types needed for a given scenario.

Other communicative act types are available to model nonverbal acts (such as gestures) and signals (e. g., to indicate that a participant has started or finished speaking). Additionally, there are the types *physical act* and *meta-act*.

The basic structure of an act is as follows:

Definition (Act type)
An *act type* is a TFS of type *Act* with the following roles:

$$\begin{bmatrix} \textit{Act} & \\ \textsc{has\_initiator} & \textit{Agent} \\ \textsc{has\_precondition}^{*} & \textit{Condition} \\ \textsc{has\_postcondition}^{*} & \textit{Condition} \\ \textsc{has\_parameter}^{*} & \textit{FormalParameter} \\ \textsc{has\_beginTime} & \textit{Time} \\ \textsc{has\_endTime} & \textit{Time} \end{bmatrix}$$

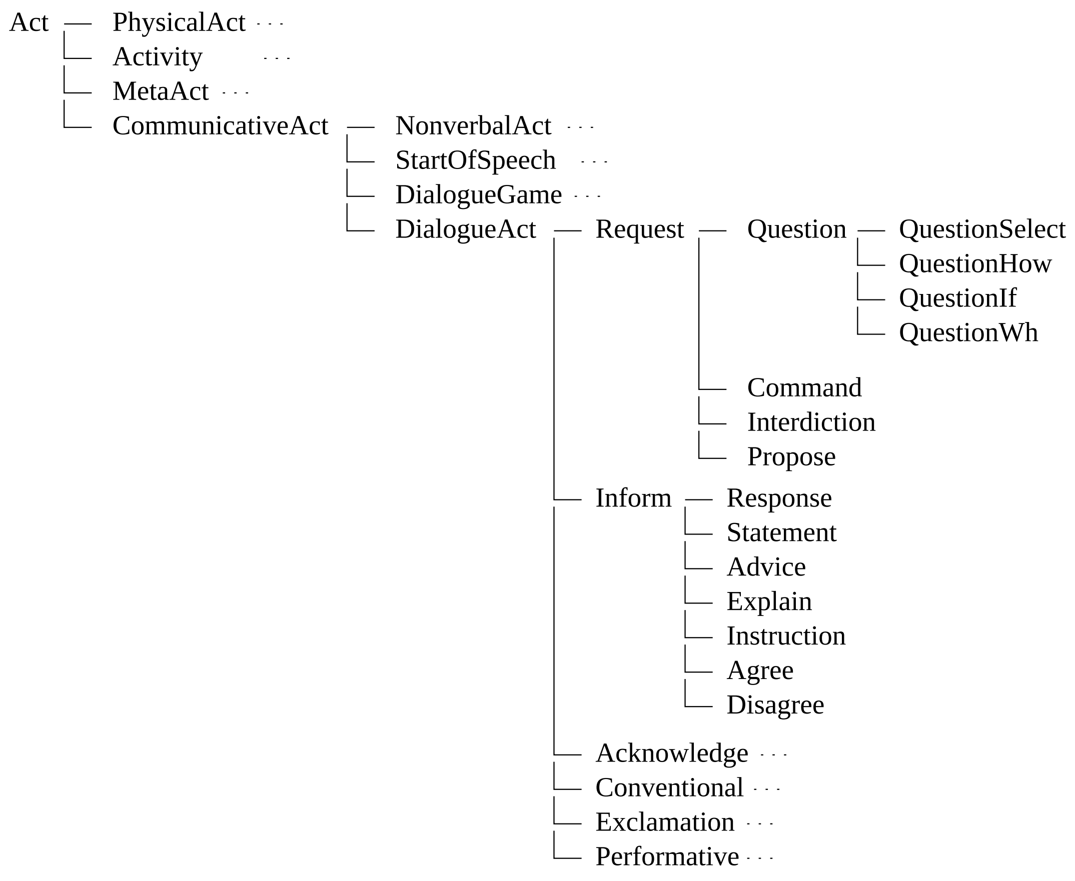


Figure 5.9: Partial tree of subcategories of *Act*

A basic act always has an initiator, but is not necessarily directed at someone else. An *addressee* slot is introduced in the *CommunicativeAct* type, and also available for the *MetaAct* type.

5.6.1.1 Examples of Dialogue Acts

Concepts inheriting from *Act* can add new roles holding the content of the act they define. For example, the *Agree* concept has an additional *has_content* role that contains the semantic representation of what was agreed to. Some simpler acts like yes/no answers or acknowledgements do not need to provide more information beyond their type. Here are some examples showing partial instances from *VirtualHuman*:

- **Greeting**

  A *Greeting* dialogue act does not necessarily need additional content. The semantics are already sufficiently clear by the type of the act alone, supplied with an initiator and a set of addressees. Depending on how utterances are generated, however, the act could also possibly be enriched by, e. g., canned text content for a surface utterance; but if it is missing, multimodal generation could also just supply a greeting gesture:

$$\begin{bmatrix} \textit{Greeting} & \\ \textsc{has\_initiator} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”} \end{bmatrix} \\ \textsc{has\_addressee} & \left\langle \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Herzog”} \end{bmatrix}, \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Kaiser”} \end{bmatrix} \right\rangle \\ \textsc{has\_cannedText} & \textit{I welcome you all to our quiz!} \end{bmatrix}$$

- **Inform**

  An *Inform* act has to specify a *has_content* role that states what the informing is about, in this case, that a football player named “Michel Platini” shoots with his right foot and scores a goal in the bottom right corner:[4]

$$\begin{bmatrix} \textit{Inform} & \\ \textsc{has\_initiator} & \begin{bmatrix} \textit{Agent} & \\ \textsc{has\_name} & \textit{“User1”} \end{bmatrix} \\ \textsc{has\_addressee} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_gender} & \textit{“female”} \end{bmatrix} \\ \textsc{has\_content} & \begin{bmatrix} \textit{Goal} & \\ \textsc{has\_agent} & \begin{bmatrix} \textit{FootballPlayer} & \\ \textsc{has\_name} & \textit{“Michel Platini”} \\ \ldots & \end{bmatrix} \\ \textsc{has\_direction} & \textit{BottomRight} \\ \textsc{has\_style} & \textit{RightFootShoot} \\ \ldots & \end{bmatrix} \\ \ldots & \end{bmatrix}$$

  This is a more sophisticated example where an utterance can be produced by a generation component doing “deep generation” from the semantic representation of the content.

  Groups of characters can be addressed by using underspecification and/or concept specialization a in the addressee slot. For example, to address all virtual characters, but not any users, the *addressee* slot is filled by an empty instance of type *Character* (specialization). The example above additionally uses underspecification to address all *female* characters (but not female users).[5]

- **Agree**

  An *Agree* act can have different types of values in his *has_content* role: A participant can (1) either explicitly agree to an *Inform* statement made by another participant, or simply (2) implicitly “agree with” another participant. If the expert Kaiser explicitly agrees with user 1’s opinion about some football action and tells the moderator as well as user 1 about it, the instance looks like this:

[4]Note that the “initiator” role in acts are of type *Agent*, since acts can be initiated by—and addressed to—characters, users, and other modules, which are all *Agents*. Where the initiator is a character, the roles are filled by the more specific subconcept *Character*.

[5]This example is not actually possible in *VirtualHuman*, since the users’ genders are not known by the system.

$$\left[\begin{array}{ll}
\textit{Agree} & \\
\textsc{has\_initiator} & \left[\begin{array}{ll}\textit{Character} & \\ \textsc{has\_name} & \textit{“Kaiser”}\end{array}\right] \\
\textsc{has\_addressee} & \left\langle \left[\begin{array}{ll}\textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”}\end{array}\right], \left[\begin{array}{ll}\textit{Character} & \\ \textsc{has\_name} & \textit{“User1”}\end{array}\right] \right\rangle \\
\textsc{has\_content} & \left[\begin{array}{ll}
\textit{Inform} & \\
\textsc{has\_initiator} & \left[\begin{array}{ll}\textit{Agent} & \\ \textsc{has\_name} & \textit{“User1”}\end{array}\right] \\
\textsc{has\_addressee} & \left[\begin{array}{ll}\textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”}\end{array}\right] \\
\textsc{has\_content} & \textit{FootballAction} \ldots
\end{array}\right]
\end{array}\right]$$

In case (2), the discourse modeler tries to determine which *Inform* act was referred to by the agreement from the conversation history and integrate it into the content of the *Agree* act.

5.6.1.2 Physical Acts

Another dimension that comes into play within a virtual performance is concerned with interactions with the environment. The agents of a virtual reality system are situated because they move and act in a simulated environment including simulated physical objects, and quasi-physical objects like menus superimposed over the scene rendition on the screen.

Carlson, in his work on dialogue games (Carlson, 1983), briefly compared the mental deliberations of a person to conducting an internal dialogue with herself. He also stated that *“Nature is in the audience of every move”*. In addition to communicative actions, we model the *physical* interactions of characters with the world in terms of a rule-based dialogue. The percepts that an agent receives after it initiated an action can be seen as an answer given by the environment to the question, “what happens if I do *this*?”, and in some cases, there may be more than one possible answer (e. g., an action can succeed or fail).

An example for a physical act in *VirtualHuman* is when the moderator moves football players to positions on the virtual playing field, which is done by producing acts of types *ZambAddPlayer*, *ZambRemovePlayer* and *ZambMovePlayer*:

$$\left[\begin{array}{ll}
\textit{ZambAddPlayer} & \\
\textsc{has\_initiator} & \left[\begin{array}{ll}\textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”}\end{array}\right] \\
\textsc{has\_player} & \left[\begin{array}{ll}\textit{FootballPlayer} & \\ \textsc{has\_name} & \textit{“Michael Ballack”}\end{array}\right] \\
\textsc{has\_fieldPosition} & \left[\begin{array}{ll}\textit{MidfieldPosition} & \\ \textsc{has\_side} & \textit{“half-right”}\end{array}\right]
\end{array}\right]$$

The value of the role *fieldPosition* in this case can be of type *FootballFieldPosition* or of type *SpatialReference*. In the latter case, the *FADE* module resolves the spatial reference, if possible, and replaces the value with a *FootballFieldPosition*.

Concept	Senders	Receivers	Purpose
SetGoal	narration engine	characters	Sets a goal for a character or group of characters, specifying an activity and role values
RetractGoal	narration engine	characters	Retracts a currently active goal
GoalFeedback	character	narration engine	Contains feedback information about a current or terminated goal
CreateCharacter	narration engine	controller	Creates and activates a character CDE
RemoveCharacter	narration engine	controller	Removes a character CDE completely from the dialogue system
Deactivate	narration engine	character	Stops processing for a character without removing it from the dialogue system
Assert	narration engine	controller, characters	Asserts an object or a relation into the ontology of a character
Reset	narration engine, controller	controller, characters	Resets the dialogue system
Expectation	characters	analysis modules	Contains information about the utterance types expected by a character, and lexicon updates
(playerML)	narration engine	player	Sends a playerML command directly to the player
(playerFeedback)	player	controller	Informs about realization state of player commands

Figure 5.10: Subconcepts of *MetaAct* (the last two acts are *VirtualHuman*-specific)

The environment also can and frequently does "take the initiative for itself" (which is called "events happening"). The interaction patterns for physical interactions do not correspond one-to-one to conversational ones, and are generally less complex (the environment will not pose rhetorical questions, for example). However, by capturing the interactions between characters, the character-internal exchanges and physical actions in the same model, we aim to be able to use the same mechanism for all three interaction types. Characters that witness the actions of others and their consequences may also, in analogy to the overhearers in a conversation, perceive the "answering" reaction of the environment, if their perception filter allows for it.

5.6.1.3 Meta-Acts

Besides communicative acts and physical acts, there is a set of *Meta-Acts*. Meta-Acts are not realized as utterances in the environment, but sent "silently" to the receiver (i. e., human users can not perceive them). They can be produced or processed by the conversation participants and/or other modules in the system. Figure 5.10 shows a table of possible Meta-Acts.

The acts initiated by the narration engine support the directing of the story in an interactive narrative. It is possible to change the scenario by adding, disabling and removing CDEs. The narration engine can also send goals directly to one or a group of CDEs, or make assertions about objects and relations that are to be integrated into the respective ontologies of the receivers of the assertion.

CDEs send Meta-Acts in return to provide feedback about the status of their goals when they are finished, or when an event occurs that the narration engine has subscribed to. If a CDE expects a dialogue act from another conversation participant, it produces an *Expectation* Meta-Act which is sent to the analysis modules. In *VirtualHuman*, the player module also sends Meta-Acts that inform about the realization state of commands (fetched, started, finished) and allow the conversation manager to synchronize actions of the characters with the presentation.

5.6.2 Dialogue Game Types

Figure 5.11 lists some dialogue game types that occur in different scene contexts of the *VirtualHuman* system. Dialogue games are a device for the participant to collaborate in communicative exchange by providing a protocol that defines which reactions can follow an action, and what it means in context.

Context	**Game Types**
Moderation	Greeting users, introducing Experts, explaining the quiz rules, alerting to timeouts, announcing different game stages, providing help
Playing the scene quiz	Requesting to select from available answers, requesting to propose an answer, commenting other's opinion
Creating the football team	Requesting to propose an action, requesting information, physically executing and evaluating player moves, hinting

Figure 5.11: Some game types occurring in different contexts in *VirtualHuman*

Using dialogue games, an agent can gain some assurance about the presumed behavior of other agents. They can be used like atomic acts to plan a sequence of actions. Unlike atomic acts, however, the outcome of a dialogue game is not wholly determined beforehand. An act in a dialogue game may often leave several alternatives of how a counterpart can continue. Additionally, other agents may choose to violate the implicit social rules of interaction, i. e., refuse to play along in the game.

A dialogue game type captures a rule-based exchange of acts. It can be depicted as a finite state automaton (FSA). In the ontology, dialogue game types are represented by *DialogueGame* instances that use *DialogueGameState*s and *DialogueGameEdge*s roles that define the automaton's graph. Figure 5.12 shows a simple FSA for an interaction that starts with a question from the game's initiator, followed by a response by the addressed participant. Like

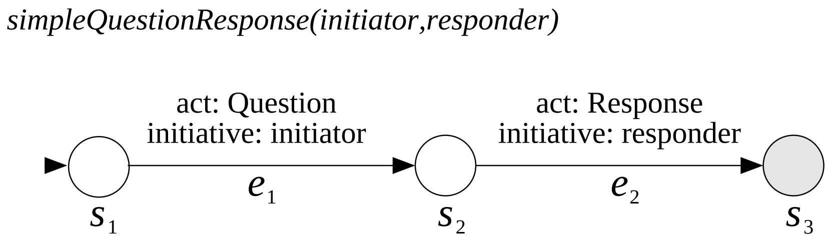


Figure 5.12: Finite state automaton showing the basic structure for a simple Question-Response dialogue game type.

acts, dialogue games have formal parameters that specify how their roles are assigned. The elementary parameters for a dialogue game are its initiator and its counterpart, the responder.[6] If it is not removed by a perception filter (either explicitly, or implicitly in case of the user), other participants will also receive the message containing the dialogue act, but be able to tell that they are not addressed directly. The simple example FSA would be represented in the ontology as shown in Figure 5.13.

A dialogue game has a unique initial state. The game is advanced by traversal of edges until one of a set of final states is reached. Each edge traversal represents the realization of one act by the participant that holds the edge's initiative.

Figure 5.14 shows a slightly more complex game, a question made by a teacher to a pupil (the darkened states are terminal). It has two final states corresponding to different outcomes. In state s_2, there are two possible edges available to the responder; if e_3 is taken, the game terminates immediately, whereas e_2 is to be followed by another edge that lets the initiator evaluate the response.

The whole game, as well as the constituent edges, can be assigned preconditions and postconditions, like activities. A game is only applicable, and the edge is only traversable, if the preconditions are satisfied, and the postconditions express what is to be expected after the termination of the game, or the traversal of the edge, respectively.

In summary, the concepts in the knowledge base that are necessary to define dialogue game types comprise dialogue games, dialogue game edges, and dialogue game states.

Definition (Dialogue Game State)
A *dialogue game state* is a concept of type *DialogueGameState*, which has the following structure:

$$\begin{bmatrix} \textit{DialogueGameState} & \\ \textsc{has\_name} & \textit{String} \end{bmatrix}$$

Definition (Dialogue Game Edge)
A *dialogue game edge* is a concept of type *DialogueGameEdge*, which has the following structure:

[6]A responder group, e. g., all experts in *VirtualHuman*, can be represented by underspecification.

$$\begin{bmatrix}
\textit{DialogueGame} & \\
\text{HAS\_NAME} & \textit{``simpleQuestionResponse''} \\
\text{HAS\_PARAMETER*} & \left\langle \begin{bmatrix} \textit{FormalParameter} & \\ \text{HAS\_NAME} & \textit{``initiator''} \\ \text{HAS\_VALUE} & \boxed{4}\ \textit{Character} \end{bmatrix}, \begin{bmatrix} \textit{FormalParameter} & \\ \text{HAS\_NAME} & \textit{``responder''} \\ \text{HAS\_VALUE} & \boxed{5}\ \textit{Character} \end{bmatrix} \right\rangle \\
\text{HAS\_INITIALSTATE} & \boxed{1} \\
\text{HAS\_FINALSTATE*} & \langle \boxed{3} \rangle \\
\text{HAS\_EDGE*} & \left\langle \begin{array}{l}
\begin{bmatrix} \textit{DialogueGameEdge} & \\ \text{HAS\_NAME} & \textit{``e1''} \\ \text{HAS\_ACT} & \begin{bmatrix} \textit{Question} & \\ \text{HAS\_INITIATOR} & \boxed{4} \\ \text{HAS\_ADDRESSEE*} & \langle \boxed{5} \rangle \end{bmatrix} \\ \text{HAS\_INITIATIVE} & \boxed{4} \\ \text{HAS\_SOURCE} & \boxed{1} \begin{bmatrix} \textit{DialogueGameState} & \\ \text{HAS\_NAME} & \textit{``s1''} \end{bmatrix} \\ \text{HAS\_TARGET} & \boxed{2} \begin{bmatrix} \textit{DialogueGameState} & \\ \text{HAS\_NAME} & \textit{``s2''} \end{bmatrix} \end{bmatrix}, \\
\begin{bmatrix} \textit{DialogueGameEdge} & \\ \text{HAS\_NAME} & \textit{``e2''} \\ \text{HAS\_ACT} & \begin{bmatrix} \textit{Response} & \\ \text{HAS\_INITIATOR} & \boxed{5} \\ \text{HAS\_ADDRESSEE*} & \langle \boxed{4} \rangle \end{bmatrix} \\ \text{HAS\_INITIATIVE} & \boxed{5} \\ \text{HAS\_SOURCE} & \boxed{2} \\ \text{HAS\_TARGET} & \boxed{3} \begin{bmatrix} \textit{DialogueGameState} & \\ \text{HAS\_NAME} & \textit{``s3''} \end{bmatrix} \end{bmatrix}
\end{array} \right\rangle
\end{bmatrix}$$

Figure 5.13: Instance of a dialogue game type corresponding to Figure 5.12.

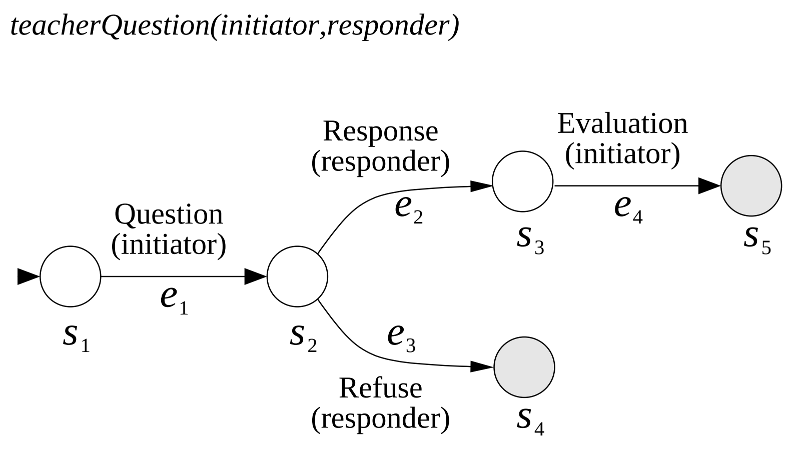


Figure 5.14: Game type FSA for a teacher's question

$$\begin{bmatrix} \mathit{DialogueGameEdge} & \\ \textsc{has\_name} & \mathit{String} \\ \textsc{has\_precondition}* & \mathit{Condition} \\ \textsc{has\_postcondition}* & \mathit{Condition} \\ \textsc{has\_parameter}* & \mathit{FormalParameter} \\ \textsc{has\_act} & \mathit{Act} \\ \textsc{has\_initiative} & \mathit{ParticipantRole} \\ \textsc{has\_sourceGameState} & \mathit{DialogueGameState} \\ \textsc{has\_targetGameState} & \mathit{DialogueGameState} \end{bmatrix}$$

A *ParticipantRole* value can either be represent an *Initiator* (who started the game) or the *Responder* counterpart.

Definition (Dialogue Game Type)
A *dialogue game type* is a concept of type *DialogueGame*, which has the following structure:

$$\begin{bmatrix} \mathit{DialogueGame} & \\ \textsc{has\_name} & \mathit{String} \\ \textsc{has\_initiator} & \mathit{Agent} \\ \textsc{has\_parameter}* & \mathit{FormalParameter} \\ \textsc{has\_precondition}* & \mathit{Condition} \\ \textsc{has\_postcondition}* & \mathit{Condition} \\ \textsc{has\_initialState} & \mathit{DialogueGameState} \\ \textsc{has\_finalState}* & \mathit{DialogueGameState} \\ \textsc{has\_edge}* & \mathit{DialogueGameEdge} \end{bmatrix}$$

This representation covers the five components of a formal description of dialogue games required by (McBurney and Parsons, 2002a,b) that were stated in Section 2.5.1.1:

1. **Commencement rules:** A dialogue game may begin if its preconditions are satisfied, and must begin in its initial state (i. e., the first move must be along an outgoing edge of the initial state)
2. **Locution rules:** All instances of act types occurring on an edge in the game may occur.
3. **Combination rules:** In the context of a given game state, the set of permitted locutions is restricted to instances of act types on those outgoing edges with satisfied preconditions.
4. **Propositional commitments:** The propositions that the participants are obliged to commit to with a given utterance move are those occurring in its postconditions, as parameterized by the game parameters and the actual act instances.
5. **Termination rules:** A dialogue game ends when it reaches any of its terminal states.

The composition operations of iteration, sequencing, parallelization, and embedding for two dialogue game graphs G and H can be expressed by combining states as shown in Figure 5.15, where new transitions and additional auxiliary states are shown in dotted lines.[7]

[7] A special case would be *overlapping* games where, e. g. a sequence where the end state of the first game is conflated with the initial one of the second; also, this could be extended to more than one leading or trailing states. It would be an interesting area for further investigation to examine how overlapping games can be automatically recognized or generated.

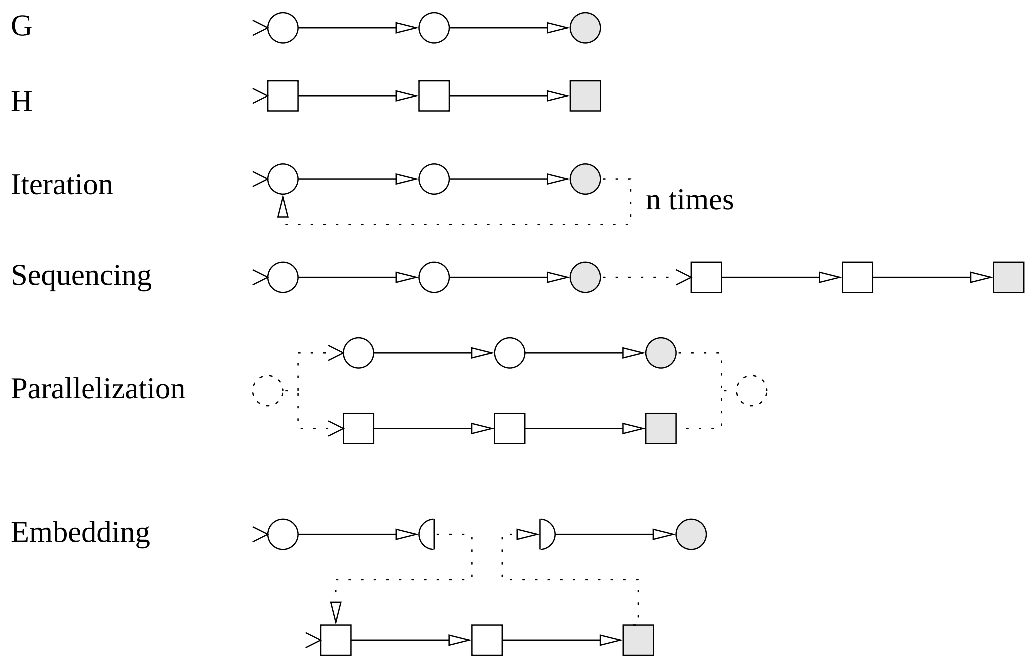


Figure 5.15: Illustration of the game composition operations

In the literature, dialogue games are sometimes understood as short initiative-response exchanges of adjacency pairs and possibly associated acknowledgment moves (e. g., (Kowtko et al., 1991). We use them to build structures that cover longer interaction spans. Arguably, there is still an upper limit to the length of sensible dialogue games.

5.6.3 Activity Types

Following (Hoc, 1988), we see an *activity* as the interaction between a subject (or more than one subject in case of joint activities) and a task, and distinguish between observable (behavioral) and non-observable (regulatory) mechanisms that make up the activity.

Figure 5.16 lists some activity types occurring in the *VirtualHuman* system. Taking up an activity means introducing an *instance* of the corresponding type into the discourse. The instances of each type are parameterized by roles specifying *who* is partaking in the activity in what function, and possibly additional parameter roles that specify *how* the activity is to proceed. An activity also usually has one or more specific conditions upon which it terminates.[8]

Like in the case of *DialogueAct*, the basic ontological concept *Activity* does not contain many roles, but its subconcepts often specify additional roles besides the participant and parameter roles. For an instance, they either have to be available from the initial knowledge bases of the participating characters, or dynamically inferred from the previous interaction. For example, successfully executing the *Lineup* activity requires knowledge about the football players enlisted in the national team roster, as well as the possible spatial positions they

[8]This is not strictly necessary; an activity can also be intended to continue either indefinitely or until it is explicitly cancelled by, e. g., a *MetaAct* from the narration module.

Activity Type	Participant Roles	Parameter Roles	Termination
Introduction	Moderator	One expert to be introduced	after completion of introduction
QuizRound	Moderator, two experts and two users	An identifier for a particular football scene in the ontologies of the virtual characters	after both users have answered the question; feedback about new scores of the users
Lineup	Moderator, expert and winning user from first round	opponent team, optional time constraints	when user states that she is finished, or the time limit has expired

Figure 5.16: Some activity types in *VirtualHuman* with their roles and goal conditions

can occupy on the football field. This information is available from the initial character ontologies. On the other hand, when a *QuizRound* activity is executed, it stores the current score dynamically in the ontology, so that it can be used in later instances to compute the resulting score.

We define

Definition (Activity Type and Activity Instance)
An *activity type* is a TFS of type *Activity* where the following roles are filled:

$$\begin{bmatrix} \textit{Activity} & \\ \textsc{has\_precondition}* & \textit{Condition} \\ \textsc{has\_postcondition}* & \textit{Condition} \\ \textsc{has\_autoRegister} & \textit{Boolean} \\ \textsc{has\_autoStart} & \textit{Boolean} \\ \textsc{has\_className} & \textit{String} \\ \textsc{has\_parameter}* & \textit{FormalParameter} \\ \textsc{has\_service}* & \textit{DialogueGame} \end{bmatrix}$$

The meanings of the roles are

- PRECONDITION – a (possibly empty) set of *Condition* instances that must be satisfied for the activity to begin,
- POSTCONDITION – a (possibly empty) set of *Condition* instances that are expected to hold after the execution of the activity,
- AUTOREGISTER and AUTOSTART – whether the activity will automatically register the dialogue games in its SERVICE slots (cf. Section 5.7.3.2), and whether it will be started when the CDE is initialized
- PARAMETER – a set of *FormalParameter* instances specifying the possible roles in the activity,
- CLASSNAME – the name of the class implementing the activity type's deliberation process type

- SERVICE – a set of service dialogue games

An activity type is underspecified, and can be used to create an *activity instance* with the same role values, which has some additional parameter roles filled. The exact set of additional roles depends on the HAS_PARAMETERS values which may reference any role present in the actual subconcept of *Activity* that is used, but include at least the following:

$$\begin{bmatrix} \textit{Activity} & \\ \textsc{has\_name} & \textit{String} \\ \textsc{has\_initiator} & \textit{Agent} \\ \textsc{has\_begin}\text{T}\textsc{ime} & \textit{Time} \\ \textsc{has\_end}\text{T}\textsc{ime} & \textit{Time} \end{bmatrix}$$

- NAME – a unique name for the activity instance, acting as an identifier,
- INITIATOR – the agent that caused the activity to be created,
- BEGINTIME – the creation time of the activity, if applicable
- ENDTIME – the termination time of the activity, if applicable

Activity is an abstract concept. For each activity known to a character, its ontology must contain a concrete subconcept of *Activity* that can be instantiated, which then yields a building block on the activity level.

For example, assume the ontology contains an instance representing an *InformationSearch*, where *InformationSearch* is a subtype of *Activity*. The concept specifies that an *Information-Search* activity has the roles of *initiator* and *addressee* of type *Character* and a *content* role that can have a value of the most general type *:THING*, as shown below:

$$\begin{bmatrix} \textit{InformationSearch} & \\ \textsc{has\_initiator} & \textit{Character} \\ \textsc{has\_addressee} & \textit{Character} \\ \textsc{has\_content} & \textit{:THING} \end{bmatrix}$$

For a CDE to offer a specific type of *InformationSearch* goal, an appropriately restricted activity type instance must be in the ACTIVITYTYPES set of the CDE, along with some information about the deliberative process type that will handle the goal, for example,

$$\begin{bmatrix} \textit{InformationSearch} & \\ \textsc{has\_content} & \begin{bmatrix} \textit{FootballPlayer} \\ \textsc{has\_nationality}\text{: } \textit{“German”} \end{bmatrix} \\ \textsc{has\_class}\text{N}\textsc{ame} & \textit{LisaProcess} \end{bmatrix}$$

When a CDE's ACTIVITYTYPES set contains this instance, it offers a goal to handle information searches about German football players.

5.7 Operation of the CDE Conversation Manager by Example

Now that the interaction building blocks are established, we look more closely at what the tasks of the conversation manager are, and how they are accomplished. For this, we already have to include some specifics about how the framework that implements the model will be structured.

Since there are many factors that contribute to the operation of the conversation manager, a top-down explanation would likely be confusing. To avoid this, we chose to describe features one-by-one as they appear while presenting an interaction example from the *VirtualHuman* system, which we introduce in Section 5.7.2. But first, we give an overview of the conversation manager components and the tasks it has to handle.

5.7.1 Tasks of the Conversation Manager

A conversation manager M has to produce the deliberative and behavioral actions of all characters and deliver the messages that communicate actions of characters, human participants, and other system modules to their destinations. This poses a variety of different sub-tasks for each part of the conversation manager:

- **Controller and Environment**

  The controller has to route and possibly process the following kinds of messages:

  - *Communicative and physical acts:*

    If a participant $I \in \mathcal{C}_M$ initiates a communicative or physical act $\textsc{Act}$ by sending a message with a set of addressed participants $A \subseteq \mathcal{C}_M$, then if $\textsc{Act}$ is an act that is to be realized in the virtual environment, a corresponding event (TTS, graphical, etc.) must be generated and passed on to the realization channels, e. g., the player output channel. This ensures that it can then be perceived by the human users, unless implicit filtering applies (cf. Section 5.4.2).

    For all character addressees $a \in A$, their explicit perception filter f_a is applied to $\textsc{Act}$, and a receives $f_a(\textsc{Act})$.

  - *Meta-acts:*

    Meta-Acts (e. g., *SetGoal*) can be addressed to one or more CDE and/or the controller. They do not have to be rendered in the environment. Like other acts, they are delivered to all addressees $a \in A$, but no perception filter is applied. If a Meta-Act is addressed to the controller itself, it can trigger action such as creating new CDEs, or suspending active ones.

- **Virtual Characters**

  Each CDE representing a virtual character must perform the following functions:

  - *Maintenance of the private world model:*

    The ontology of the character needs to be kept in sync with the environment and the internal deliberations. This includes adding and modifying instances. The character must also be able to accept *Assert* acts that manipulate its world model, e. g. from a narration engine, and make the corresponding updates.

  - *Accepting goals:*

    For all goals they offer, the characters have to accept *SetGoal* meta-acts, create and adopt corresponding goal instances, and upon termination of a goal, return feedback messages informing about the outcome.
  - *Producing goal-directed behavior:*

    When a goal has been adopted, a character has to determine and initiate communicative and other actions that lead towards its fulfillment.
  - *Consuming and interpreting communicative acts:*

    When a character receives communicative acts, it has to interpret them with respect to its running activities or adopt new activities to deal with them, and engage in the corresponding dialogue games.

- **External Interactions**
  - *User CDEs:*

    The user CDEs are connected to the input channels delivering user input and they route the corresponding utterances to the addressed character CDEs.
  - *External Applications and Modules, Proxy CDEs*

    Since a dialogue system is often used as an interface to the functionality of external applications, e. g., databases, there must be a way to interact with them. In this case, the conversation manager acts as a client to a module that provides an API to the application. The communication with the application is then routed through input and output channels that convert dialogue acts to messages that can be processed by the API.

    Depending on the setup, a *proxy CDE* for an application or module may be used. A proxy CDE partakes in the interaction like the virtual characters, but is not rendered in the scene. This can be appropriate if the application or module acts like another person participating in the conversation. One case is the narrative engine in *VirtualHuman*, which can be seen as impersonating a director of a play.

  User and proxy CDEs encapsulate the module functionality. No other CDEs need to subscribe to the channels delivering application resp. user input. There are additional possibilities for debugging and testing. For example, the module or the user can be replaced by a simulation, or responses can be provided manually (see Sections 7.2.3 and 7.2.2).

Instead of treating each of the aspects separately, we use an example that illustrates how they come into play in a real interaction.

5.7.2 The Running Example from *VirtualHuman*

Figure 5.18 is an excerpt from a quiz interaction in the *VirtualHuman* scenario and illustrates a number of features. Throughout the following sections, we will refer to it as a running example. Involved as participants are two human users, the virtual moderator, and two competing virtual football experts, Miss Herzog and Mr Kaiser, who—by design of the narrative—have an aversion to each other.

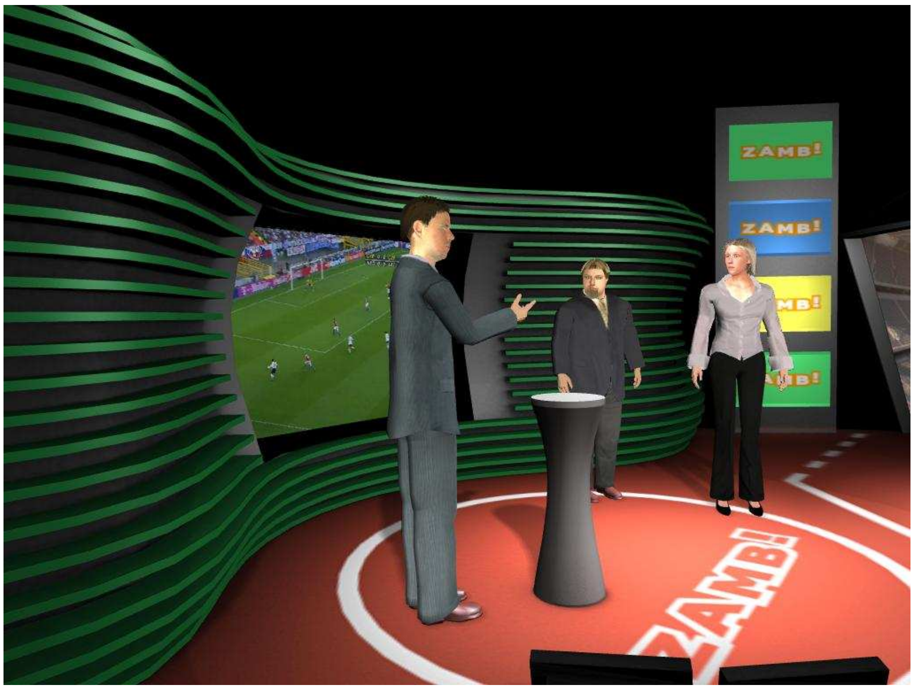


Figure 5.17: A scene from the quiz game: both experts listening to the moderator

For the moderator character, the purpose of the underlying activity of type *QuizRound* is to present a football scene, to get both human users in turn to make a guess about the probable outcome of the scene, and finally to assign them scores depending on whether their guesses were correct. The role of the experts in this activity is to answer questions from the moderator or the human users in order to give them advice (the excerpt does not cover a whole instance of this goal). The implicit goal of the users is to score points with correct answers. The result of the activity is the score. It is returned back to the narration engine on termination, which can adapt the story accordingly (cf. Section 7.2.2).

5.7.3 The Activities Available to a Character

The actions that a virtual character is able to perform can be grouped according to different criteria. One is the type of interaction pattern. In a task-oriented system, one can consider information-retrieval actions, commands, and answering system questions as re-occurring patterns. Another possibility is to treat actions relating to the same application as related. The first categorization is based on similarities between the patterns: question-answer exchanges proceed in much the same way across different applications. The second categorization emphasizes that actions occurring in the same application are related thematically, and may also share common data and interaction context.

In our model, both categorization schemes are used, but they apply to different layers. *Activities* represent processes that are thematically related, e. g., by virtue of belonging to the

			illustrates
	DIRECTOR:	*(goal start)*	initial setup of the conversation manager, explicit creation of activities for external goals
(1)	MODERATOR:	*Now for an interesting scene* [shows video on screen]	using the private knowledge, deliberation and action planning, single-initiative dialogue games, physical action
(2)	MODERATOR:	*What happens next?* [↗ counting gesture] *One – Ballack scores the goal,* [↗ counting gesture] *Two – the keeper does a parade,* [↗ counting gesture] *Three – Ballack kicks the ball into the sky.*	generating multimodal utterances from semantic descriptions
(3)	MODERATOR:	*What is your guess, Mister Kaiser?*	implicit creation of activities
(4)	EXPERT KAISER:	[smiles] *I believe that Ballack scores the goal.*	active role of expert
(6)	MODERATOR:	*Spoken like a real football trainer!*	"canned" response
(7)	MODERATOR:	*Now it is your turn, [↗ pointing gesture] player one. Make your guess!*	
(8)	USER 1:	*I think Mister Kaiser is right.*	using discourse references
(9)	MODERATOR:	*Alright, answer one.*	resolving references
(10)	EXPERT HERZOG:	[angry] *How can you trust such an amateur!* [Mr Kaiser smiles]	affective response, spontaneous character interjection
(11)	MODERATOR:	*Now, player two, what do you think?*	
(12)	USER 2:	*Moderator, what do you think?*	Explictly addressing characters, initiating subgame
(13)	MODERATOR:	*Sorry,* [shrugs] *I'm not allowed to help you; please ask one of the experts. Your answer?*	passive role of moderator, continuing interrupted game
(14)	USER 2:	*Answer two.*	answering directly
(15)	MODERATOR	*We will see whether that is correct. Lets look at the end of the situation* [shows second part of video]	addressing all other participants as a group
(16)	MODERATOR:	*The answer of user one was correct. Unfortunately, user two has guessed wrong.*	evaluation, goal completion and feedback
	DIRECTOR:	*[*gets successful goal feedback with score information [the moderator's activity context is updated with the new score]	

Figure 5.18: Running example: Excerpt from a quiz dialogue in *VirtualHuman*

same application or goal, and that may include a series of interactions over a longer part of the whole conversation. A character that is answering to a question by the user is performing a specific activity that represents, e. g., the football quiz application in *VirtualHuman*, and may use a generalized *Request-Response* interaction pattern that also occurs in other applications to accomplish one part of the activity.

New activity instances for a CDE can be created explicitly by a message from an external module that exerts control over the CDEs, such as the narration engine, or as a subgoal of an already running activity. It can also be created implicitly because it offers a *service* in response to a dialogue act by another conversation participant. The activity types supported by a given CDE are the ones in its ACTIVITYTYPES set, the registered services are the domain of the *registeredServices* function of the CDE.

The *QuizRound* activity of the running example is an external goal; the narration engine schedules and triggers it as a part of the overall story of the scenario. When it begins, each of the involved virtual characters (the moderator and the two experts) needs to create a new instance of the activity type *QuizRound* with parameters supplied by the narration engine.

5.7.3.1 Explicit Creation of an Activity Instance

$$\begin{bmatrix} \textit{SetGoal} & \\ \textsc{has\_initiator} & \begin{bmatrix} \textit{Agent} & \\ \textsc{has\_name} & \textit{“Director”} \end{bmatrix} \\ \textsc{has\_addressee} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”} \end{bmatrix} \\ \textsc{has\_content} & \begin{bmatrix} \textit{QuizRound} & \\ \textsc{has\_moderator} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”} \end{bmatrix} \\ \textsc{has\_contestant} & \left\langle \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“User1”} \end{bmatrix}, \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“User2”} \end{bmatrix} \right\rangle \\ \textsc{has\_expert} & \left\langle \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Kaiser”} \end{bmatrix}, \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Herzog”} \end{bmatrix} \right\rangle \\ \textsc{has\_quizQuestion} & \begin{bmatrix} \textit{FootballQuizQuestion} & \\ \textsc{has\_name} & \textit{“WM1998-france-brazil-petit”} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 5.19: *SetGoal* message from the narration engine

A new activity instance is created explicitly if the CDE receives a *SetGoal* act that can either originate from another module in the dialogue system, or from another activity of the same CDE. *SetGoal* is a meta-act (see Section 5.6.1.3) and specifies how the activity type should be instantiated; its *content* slot contains an activity type instance with the necessary parameters. At the beginning of the running example, the *Moderator* CDE receives a message from the narration engine via the *Director* CDE (as a proxy for the narration engine) that contains a *SetGoal* instance like in Figure 5.19.

From the value of the *content* role it is clear that the character named “Moderator” is to take the moderator role of the *QuizRound*. Note that in this particular case, the other participants—

virtual characters taking the expert roles and human contestants—do *not* receive a similar goal. They will partake in the activity implicitly by reacting to actions of the moderator or the users, which will be explained in the next section.

The CDE that receives the explicit goal G then needs to find an activity type to instantiate. This is done by finding an element $a \in$ ACTIVITIES, where a is a strict best match for the *has_content* element of G (if no such element can be found, no activity can be adopted, and the goal has to fail instantly). G and A are then unified, and the resulting TFS is used as the activity instance. The activity is placed as a child of the root activity of the CDE and starts by determining what action is necessary to complete the goal.

5.7.3.2 Implicit Creation of an Activity Instance

The second way an activity instance can be created is implicitly. This happens when a dialogue act A is received that is not currently expected by a running activity, and which matches one of the services of the CDE. Recall from Section 5.4.1 that a CDE has a function *services* which maps from act types to activity types. The purpose of this function is to find an activity that can deal with the dialogue act. The CDE finds the element A' that is the strict best match for A in the domain of *services*; then, *services*(A') is the activity type that is taken to handle A.

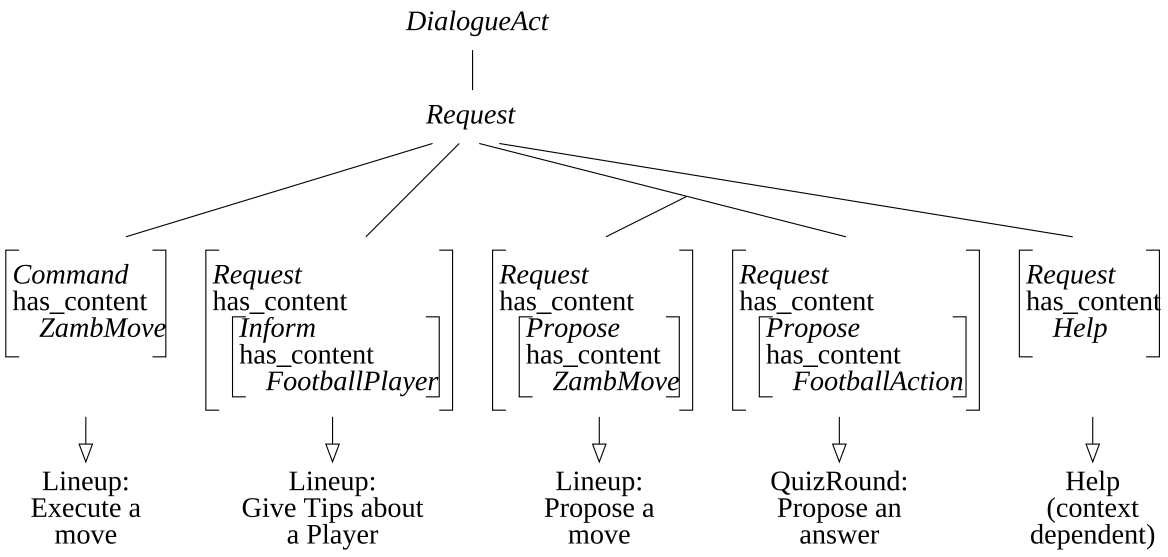

DialogueAct
Request
Command has_content ZambMove
Request has_content Inform has_content FootballPlayer
Request has_content Propose has_content ZambMove
Request has_content Propose has_content FootballAction
Request has_content Help
Lineup: Execute a move
Lineup: Give Tips about a Player
Lineup: Propose a move
QuizRound: Propose an answer
Help (context dependent)

Figure 5.20: Some service mappings of the expert characters in *VirtualHuman*

Figure 5.20 shows some of the service mappings for the expert characters in *VirtualHuman*. The five TFS shown are all instances of the *Request* type, or subtypes thereof (*Command* is a subconcept of *Request*), and they are all elements from the domain of the *services* function of an expert CDE.

In the example, the human user asks one of the experts for his opinion in turn (3):

(3)	USER 1:	*What do you think, Mister Kaiser?*
(4)	EXPERT KAISER:	[smiles] *I believe that Ballack scores the goal.*

The resulting dialogue act received by the expert's CDE is a strict best match for the second-to-last TFS in Figure 5.20, which is mapped to the *QuizRound* activity type by the *services* function. At this point, no such activity is running for the expert (because, as we said in the last section, a *QuizRound* goal was only sent to the moderator CDE), so a new activity instance of that type is started in the expert's CDE that subsequently has to deal with the contribution.

5.7.4 Deliberation in Activities

Contrary to dialogue acts, activities (and also dialogue games) are not atomic messages, but *executables* that run over a period of time in a CDE. During the execution of an activity, different kinds of deliberation can be performed: (1) adopting intentions in the form of dialogue games for one's own initiatives to satisfy the goal, and executing them; (2) processing of contributions of other participants, and integrating them with running dialogue games; and (3) other general computations on the knowledge base, including drawing task-related inferences.

This means that activities have to be programmable. The model itself does not prescribe how the executable bits of an activity are programmed, but defines a black-box protocol on their input and output:

(1) An executable can access the ontology of the character, all acts that pass the perception filter, and the *SetGoal* message that contains the activity's parameters (if the activity was started explicitly) as input,

(2) it can modify the contents of the ontology, execute dialogue games, start sub-activities, and call external modules (such as a planning algorithm), and

(3) when an activity that was started explicitly terminates, it must send a *GoalFeedback* message to the initiator of the *SetGoal* message that contains information about its success state.

The rationale for the black-box approach is that it offers great flexibility in choosing different paradigms to implement deliberation, and the possibility to use different ones in the same system. Chapter 6 describes the different paradigms that were used to implement different kinds of executables.

The deliberations that the moderator's CDE has to perform in the example immediately after the *QuizRound* activity has been started are of the first kind. The moderator initially has to examine his knowledge base to determine more information, since the *SetGoal* message does not contain all information needed for the performance of the activity: the given *FootballQuizQuestion* is only an underspecified instance. In particular, it only contains the *name* of the quiz round, but not the video to be shown, question to be asked, or the possible answers. If information is missing, it has to be obtained by finding an instance in the ontology that best matches the provided slots.

This best match is the one *FootballQuizQuestion* instance in the moderator's ontology—part of its "expert knowledge"—that has the same *name* and provides the additional slots that are necessary to perform the quiz question. Figure 5.21 shows an example (the *FootballAction*

$$\begin{bmatrix} \textit{FootballQuizQuestion} & \\ \textsc{has\_name} & \textit{“WM1998-france-brazil-petit”} \\ \textsc{has\_videoDescription} & \begin{bmatrix} \textit{VideoDescription} & \\ \textsc{has\_fileName} & \textit{“franceBrazil98.avi”} \\ \textsc{has\_clipLength} & \textit{18000} \\ \textsc{has\_stopFrame} & \textit{12400} \end{bmatrix} \\ \textsc{has\_answer*} & \left\langle \begin{matrix} \begin{bmatrix} \textit{Response} & \\ \textsc{has\_content} & [\textit{FootballAction}\ldots] \end{bmatrix}, \boxed{1}\begin{bmatrix} \textit{Response} & \\ \textsc{has\_content} & [\textit{FootballAction}\ldots] \end{bmatrix}, \\ \begin{bmatrix} \textit{Response} & \\ \textsc{has\_content} & [\textit{FootballAction}\ldots] \end{bmatrix} & \end{matrix} \right\rangle \\ \textsc{has\_correctAnswer} & \boxed{1} \end{bmatrix}$$

Figure 5.21: A *FootballQuizQuestion* instance in the moderator's ontology that matches the *SetGoal* of Figure 5.19

instances in the *Responses* that are abbreviated here are fully specified in the ontology and contain a semantic description of goals, fouls, etc). After a matching instance is found, it is overlayed on the *quizRound* slot of the *QuizRound* activity. The narration engine only needs to know the names, not the details of the quiz questions to trigger them.

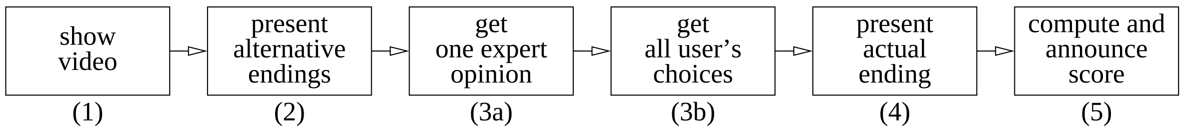


Figure 5.22: Outline of the moderator's "plan recipe" for a *QuizRound*

After the activity's parameters have been enhanced with information from the knowledge base, the moderator CDE takes steps towards completion of the goal. Figure 5.22 shows a rough "plan recipe" for a successful *QuizRound*: the moderator must present the first part of a football scene, give the choice of several alternative endings for the scene, then get the opinion of one of the experts. Afterwards, the moderator requests the guesses of both human contestants, shows the rest of the scene, and then evaluates the answers to get the new score.

The execution is not entirely straightforward. For one, the steps (3a) and (3b) also require actions from other participants, who might not be cooperative at all times. In the example, the user does not answer the question right away, but instead consults one of the experts first and agrees to his statement. The moderator's action planning mechanism must be flexible enough to accommodate this. Also, the steps (3a) and (3b) of the recipe involve very similar actions; so it is sensible to have re-usable building blocks with different parameters for each.

5.7.5 Single-Initiative Dialogue Games

Dialogue games where only one participant has the initiative act as partial recipes for an activity. However, this does not necessarily mean that only the initiator has to do any processing related to the game.

5.7.5.1 Directly Realizable Acts

The quiz turn starts with the “PresentVideo” game. It comprises two actions that involve only the moderator character, and do not require cooperation by the other participants. First the moderator shows the video, which is a physical act, and accompanies it with a comment. The moderator executes a simple dialogue game that has just two transitions. The first realizes a *PlayVideo* physical act, and the second an *Explain* dialogue act.

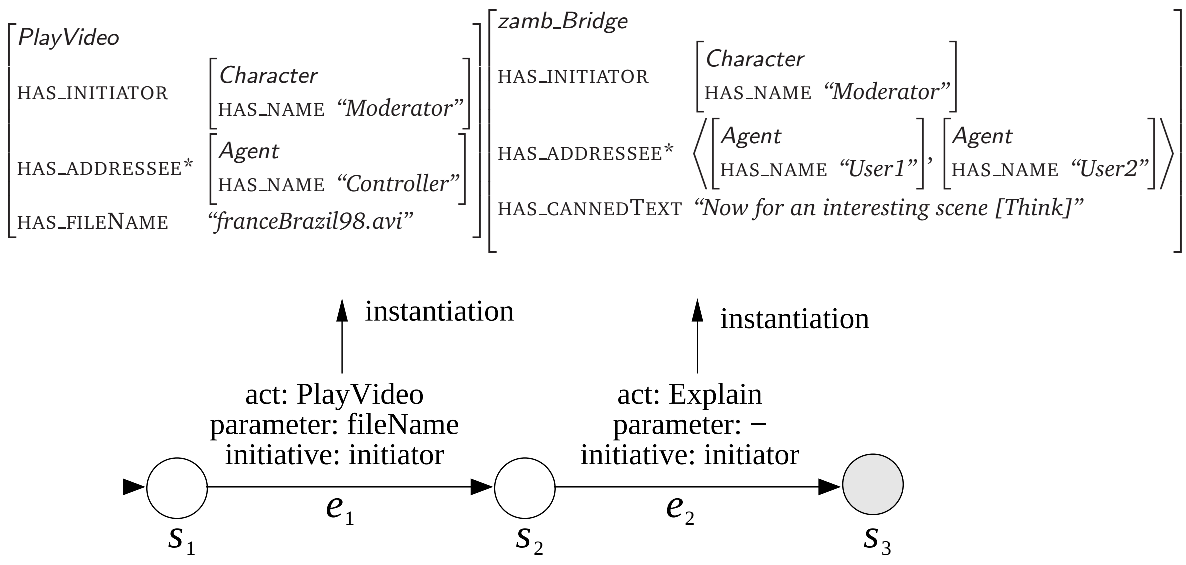
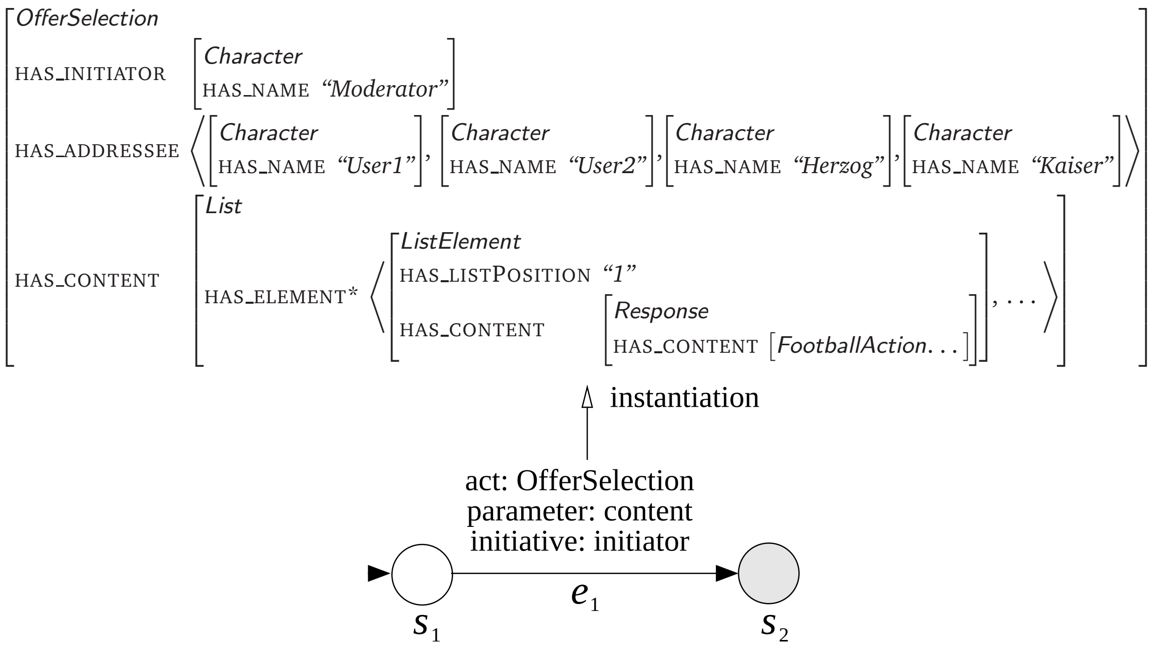


Figure 5.23: The “PresentVideo” game

The first act has one parameter, the name of the video, which can be extracted from the path *videoDescription:fileName* in the *FootballQuizQuestion* instance (Figure 5.23). Physical acts are not addressed to any other character, but are instead sent to the controller, which manages the environment (other character may however take a role in an action, e. g., as a target of a gaze). In this case, the controller forwards the *PlayVideo* act to the player channel, which will translate it to a message for the 3D player that triggers the video. Also, the act is forwarded to all CDEs that are able to perceive it (which is determined by their perception filter).

The second act is addressed to the two users, and contains the explanation for the video. In this case, an instance of *zamb_Bridge* (a subconcept of *Explain*) is used. To avoid repetitions in consecutive quiz turns, the moderator’s ontology contains a selection of *zamb_Bridge* instances from which one is randomly selected. These instances have a slot that holds a set of variants as canned text. The use of canned text is appropriate in this case, since the comments do not refer to semantic content of the videos that are presented. The annotation *[Think]* placed within the canned text is a *gesture tag* representing a class of gestures in a gesture lexicon (*gesticon*) of *VirtualHuman*as alternatives to represent “thinking”. For example, a gesture of the character scratching its head could be selected. When the act is sent to the multimodal generator, it produces a corresponding gesture at that point of the utterance.

5.7.5.2 Generated Acts



object	**effect**
game	*selection* is defined for all participants
s_2	*selection* is set to the value of *content* for all participants

Figure 5.24: Offering a selection of answers for the quiz question

The utterance that follows also involves only actions from the moderator, but it now comprises a communicative act that is addressed at all participants, including the expert characters. The information is derived from the instance describing the whole *FootballQuizQuestion* from Figure 5.21, which also contains a set of possible answers to the question about how the scene will continue. The moderator uses this information to construct an *OfferSelection* act that holds an ordered list of possibilities, as shown in Figure 5.24.

For this utterance, the content is specified using a semantic representation rather than pre-compiled canned text. The communicative act is sent to the multimodal generator, which produces a textual representation of the semantic content of the offer and the accompanying pointing gestures, as described in (Kempe et al., 2005). The result of the generation process is also text, possibly annotated with gesture tags. It can be passed on via the TTS engine to the multimodal player, which results in utterance (2):

(2) MODERATOR: *What happens next?* [↗ counting gesture] *One – Ballack scores the goal,* [↗ counting gesture] *Two – the keeper does a parade,* [↗ counting gesture] *Three – Ballack kicks the ball into the sky*

The act in its semantic form is also delivered to the addressed characters, and both expert characters will now try to make sense of the content, which we call *consuming* the utterance.

5.7.6 Consuming Acts

5.7.6.1 Determining the Processing Layer

When a CDE perceives an act initiated by another conversation participant, it has to decide whether it will react to it, and if it reacts, what processing layer it belongs to. We leverage the expectations of the characters to avoid having to do plan recognition, which is computationally very expensive (Carberry, 2001).

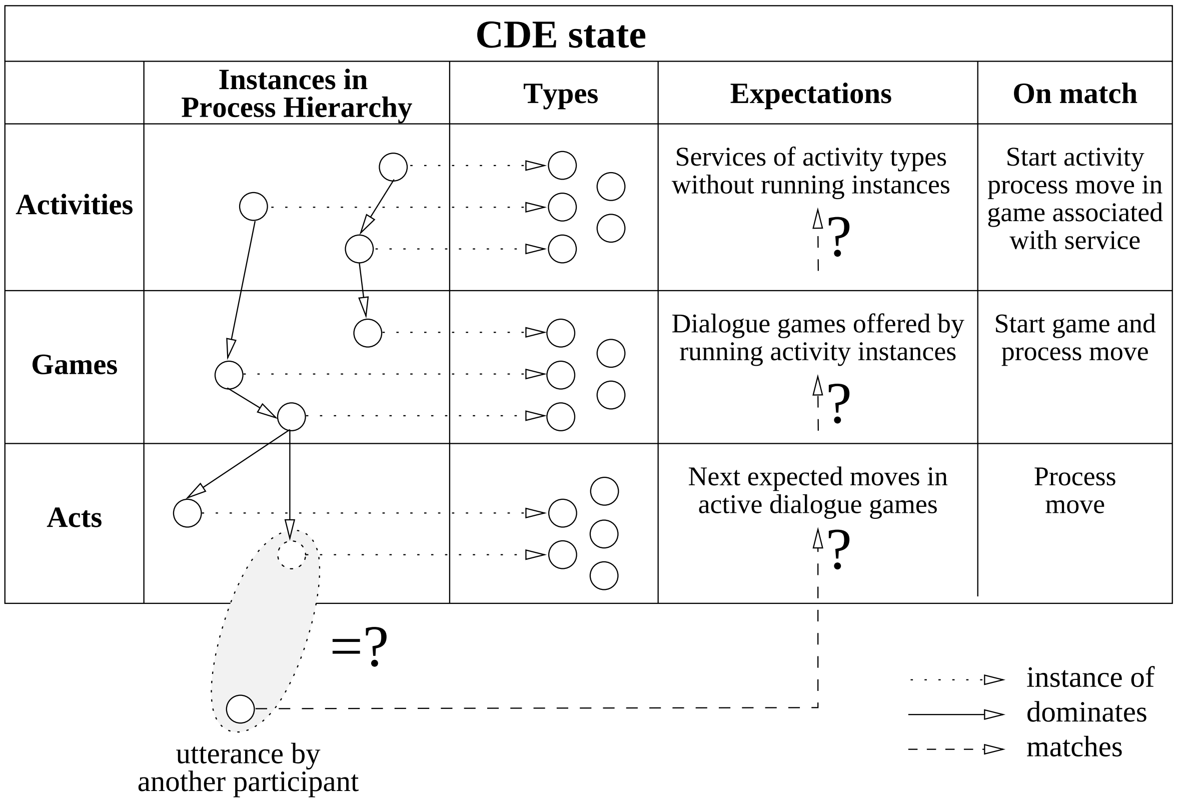

CDE state
Instances in Process Hierarchy
Types
Expectations
On match
Activities
Services of activity types without running instances
?
Start activity process move in game associated with service
Games
Dialogue games offered by running activity instances
?
Start game and process move
Acts
Next expected moves in active dialogue games
?
Process move
=?
utterance by another participant
instance of
dominates
matches

Figure 5.25: Processing utterances from other conversation participants: Finding the correct processing level by matching against expected moves, games, and services

Figure 5.25 illustrates how an utterance can be integrated given a current state of a CDE with several running activities and subactivities on the activity layer, dialogue games and subgames on the dialogue game layer, recent acts, and types for the different building blocks available in the CDE's ontology. The fundamental rule is: if an act fits an expected move in a currently active dialogue game, it will be processed in that game. An expected move in a dialogue game is one that matches the act belonging to a transition outgoing from the current state the game is in, and whose preconditions are fulfilled. Otherwise, if the act is a possible initial move in a dialogue game of a currently running activity, it is sent to this activity which will start that game. When the first two cases do not apply, the service mapping is used to determine whether there is a type of activity to deal with the act. More precisely, the method to consume an act ACT by an initiator A perceived by receiver CDE B is described by the following cases:

(1) If B is an addressee of ACT, and ACT is an expected reaction to own acts of B, e. g., previous moves in a dialogue game with A, B will continue that game by using ACT as the next move. Therefore, it will first be checked whether a move is a possible continuation or a possible subgame to currently active games. If this is the case, ACT will be processed on the dialogue game layer.[9]

(2) If B is an addressee of ACT and ACT is not a reaction, but an initiative by A that does not fit the criteria in case 1, ACT has to be processed on the activity layer. Depending on ACT, there are three further sub-cases:

  (a) ACT occurs in the context of a currently running activity of B, i. e., it matches elements of the set of services available from B's running activities. ACT will then be used to start a dialogue game in the running activity whose services offer the best match to ACT.

  (b) ACT can be interpreted in the context of an available activity type of B that does not have a running instance. This means that there is a service offered by a activity type that matches ACT best. A new activity of that type will be instantiated to handle the act.

  (c) otherwise, B is not able to interpret ACT at all. This can be because of a misunderstanding or other reasons. In this case, ACT cannot be handled by B. This case can and should be eliminated by including a "catch-all" activity that offers the service to deal with all acts of type *DialogueAct*. The catch-all activity can give feedback about the failure to A to enable A to clarify the intention. It could e. g., produce an utterance to the effect of *"Sorry, I did not understand you"*.

(3) B is not one of the addressees of ACT. In this case,

  (a) if B is interested in the content of ACT, it can use it to update its knowledge base or start an initiative of its own,

  (b) otherwise, B can choose to ignore ACT.

The state of a CDE's expectations in a given situation can also be exploited to aid input interpretation modules to resolve ambiguities in a user's input. How this is done in our framework is covered in Section 6.3.2.

In the example turn (2), both expert characters are addressed, but the act does not match any currently running dialogue game, and they are not executing an activity related to the quiz at the time. However, both have an activity type *QuestionAnswerProcess* that lets them answer questions about items in their knowledge base. It offers a service that also lets them play the *OfferSelection* game. The mapping is

[9]Case (1) can also apply in a slightly different way if B attempts to "steal" the dialogue role from the original dialogue game partner of A. A can then choose to accept or decline the role change. This is modeled in *VirtualHuman* to account for the situation where a human player tries to answer a question from the moderator character that was really addressed to the other human (see Section 1.4.1).

$$[\mathit{OfferSelection}] \mapsto \mathit{QuestionAnswerProcess}$$

The *consume* case that applies to the utterance, therefore, is case (2)(b). Both expert CDEs process the act on the activity level and start a *QuestionAnswerProcess*. This kind of process does not terminate once it is started (i. e., the termination condition is never fulfilled).

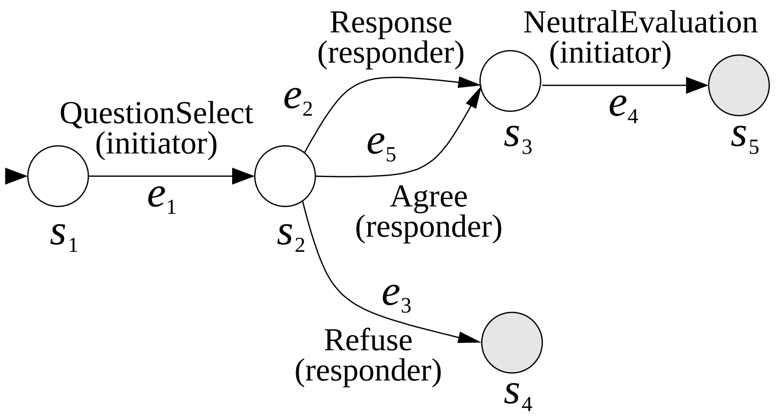
5.7.6.2 Processing an Act on the Activity Layer

When it is started, *QuestionAnswerProcess* registers (i. e., adds to the *registeredServices* function of the CDE) a service to deal with *OfferSelection* acts, because questions may actually be requests to choose from a selection of answers earlier in the conversation. Therefore, the process, upon receiving an selection offer, stores it in a local variable *selection*, in case it is referenced later. Such *task-oriented* references (i. e., ones that refer to items in the knowledge base that hold task information) are handled by the conversation manager, while *discourse-oriented* references (pertaining to the discourse history, e. g., turn (7)) are resolved by the discourse modeler.

5.7.7 Multi-Initiative Dialogue Games

In turn (3), the moderator starts the first dialogue game that also requires cooperative action from other participants:

(3) MODERATOR: *What is your guess, Mister Kaiser?*



object	precondition
game	*selection* is defined
e_2	*responder* is not moderator
e_3	*responder* is moderator
e_5	*responder* is user, *initiator* of agreed-to content is not a user

Figure 5.26: The “QuestionSelection” dialogue game

This game, called "QuestionSelection", is shown in Figure 5.26. It is derived from the "teacher's question" game shown earlier (figure 5.14). The act associated with transition e_1 is restricted to be a *QuestionSelect* (a subconcept of *Question*), and a new transition, e_5, is introduced, which adds the possibility to answer indirectly by *Agree*ing with some other statement. Also, the *Evaluation* act in e_4 is restricted to be of type *NeutralEvaluation*. The moderator is the initiator of the game, and the expert character "Kaiser" is the addressee of *QuestionSelect* (and therefore takes on the responder role). Like with turn (2), the expert has to find an activity and game corresponding to the moderator's initiative. The *QuestionAnswerProcess* activity offers this "QuestionSelection" game as a service. Since an instance of it is was started in the previous turn and is still running, case 2(a) of the processing layer determination algorithm applies and the game is also started for the expert.

Both participants start the game via transition e_1 with a *QuestionSelect* act. However, the moderator takes the initiative generating the move, which is in turn consumed by the expert. Afterwards, the game is in state s_2 for both participants. All outgoing transitions for state s_2 prescribe that the responder—the expert—has the initiative and must select one of its outgoing transitions. As can be seen from the table of preconditions in the figure, two of the transitions have preconditions that are not fulfilled. The reason for this is is that the same *QuestionSelection* game type is shared between two different participants—the moderator and the expert—who should exhibit differing behavior in the scenario. For example, if the user poses a *QuestionSelect* to the moderator, he should not voice his own opinion on the situation. the *Refuse* act in this case produces an utterance like

MODERATOR: *I cannot help you, but you could ask one of the experts instead.*

The experts, on the other hand, can use a *Response*, but are not allowed to react with an *Agree* to a previously uttered answer, since their role in the scenario is that they compete with each other and "stand their own ground", so they will as a matter of principle have their own opinions. They also must not refuse to answer. Only a user has the choice to either explicitly answer the question, or to agree to an already given answer. So in turn (4), expert Kaiser gives a direct answer:

(4) KAISER: [smiles][10] *I believe that Ballack scores the goal.*

The answer is selected at random the answers available from the *QuizRound* instance.

The next turn (6) is again executed by the moderator. It is a short evaluation of the expert's choice which is, similar to turn (1), chosen from a set of instances in the ontology. The instances used are of a concept *NeutralEvaluation* and contain canned texts like *"interesting choice!"* or *"we'll see whether you are right"*. There are also *PositiveEvaluation* and *NegativeEvaluation* concepts for use in the second game phase; however, the moderator remains neutral in this situation and does not rate the answers before showing the result video.

Beginning with turn (7), the moderator executes the *QuestionSelect* dialogue game again with different parameters to get the answers of the human quiz participants.

(7) MODERATOR: *Now it is your turn, [↗ pointing gesture] player one. Make your guess!*
(8) USER 1: *I think Mister Kaiser is right.*
(9) MODERATOR: *Alright, answer one.*

[10]Section 7.2.5 describes how the affective reaction is generated here.

$$\begin{bmatrix} \textit{Agree} & \\ \textsc{has\_initiator} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“User1”} \end{bmatrix} \\ \textsc{has\_addressee} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”} \end{bmatrix} \\ \textsc{has\_content} & \begin{bmatrix} \textit{Response} & \\ \textsc{has\_initiator} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Kaiser”} \end{bmatrix} \\ \textsc{has\_addressee} & \begin{bmatrix} \textit{Character} & \\ \textsc{has\_name} & \textit{“Moderator”} \end{bmatrix} \\ \textsc{has\_content} & \begin{bmatrix} \textit{FootballAction} \\ \ldots \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 5.27: The user *Agree*s with the opinion of expert Kaiser

The user agrees to the previous utterance of the expert, which is a possible move given the preconditions of the dialogue game. The discourse modeler *FADE* is able to reconstruct the utterance that was agreed to from the discourse history and the expectations of the conversation manager (see Section 6.3.2). This results in the enriched dialogue act shown in Figure 5.27 delivered to the moderator's CDE. The moderator can extract the same information as if the user had formulated the response himself.

5.7.7.1 Starting an Intermediate Game

A different possibility for the user is to ask the moderator (or an expert) for an opinion:

(12)	USER 2:	*Moderator, what do you think?*
(13)	MODERATOR:	*Sorry,* [shrugs] *I'm not allowed to help you; please ask one of the experts. Your answer?*

The user's comment does not fit the currently active dialogue game (in which the moderator expects an answer), and so would start a new one where the user is in the initiator role. The moderator's game is suspended while the expert answers, and continues after the subgame is completed.

5.8 Summary

This chapter introduced the model for interactions between multiple virtual characters and human users. The operation of the model was illustrated by way of an example from the *VirtualHuman* system.

The model comprises three layers: dialogue acts, which encode atomic units of communication, dialogue games, which provide for exchanges of dialogue acts between participants that are regulated by shared rules of social commitment and allow the participants to make predictions about the behavior of others, and the level of activities. The dialogue games offer a means to coordinate joint actions of the participants, and activities that represent complex behavior and can be used to implement the reasoning necessary to address related goals in an application, and to achieve them by executing a combination of dialogue games.

For further reasearch, an interesting property of the model is that it is possible to create multi-party scenarios with participants in multiple roles. An example would be a court-room situation that has multiple interesting roles (judge, plaintiff, advocate) and group roles (witnesses, jury). Formalized interactions, such as a court event, also exhibit formalized rules for items interaction that are suitable for a representation via dialogue games (e. g., an interrogation or a defense) and planning (courtroom sessions have to follow a well-defined procedure, differing by legislation). In modeling such a scenario, one such role can be taken by a human, to experience their viewpoint. For tutorial applications, this could be a useful feature, especially if the user could change dynamically between roles.

The next chapter describes how a generic CDE framework realizes this model in a way that is adaptable to a broad range of actual applications.

Chapter 6

Realization of a Conversational Behavior Generation Framework

6.1 Introduction

This chapter describes the framework that can be used to define and run applications that conform to the conversation model. It has been implemented as a set of application-independent core modules that offer the basic functionality for a conversation manager. In most cases, the core modules have to be supplemented by additional software for application-specific tasks; the framework implementation contains several APIs to facilitate these extensions. The framework also features a lightweight API for accessing and manipulating the ontological knowledge base that comprises the base ontology, together with necessary additions to capture application-specific domain and task knowledge.

The framework builds on the architecture used in the *SmartKom* system and extends it for multi-party conversation. This is facilitated by the fact that the *SmartKom* dialogue manager already provides for interleaved mixed-initiative communication with the user and the various applications via a uniform dialogue game mechanism. In effect, carrying out a multi-step task for the user in *SmartKom* generally involves having interspersed sub-conversations with one or more applications, which are—like the dialogues between the system and the user—also in terms of, e. g., questions, answers, and commands, and has some similarities to conducting a conversation with multiple communication partners.

The CDE framework has been designed to be able to accommodate dialogue system setups for different purposes. It supports an arbitrary number of human and virtual conversation participants, different reasoning mechanisms, and makes it easy to incorporate special-purpose modules to provide additional functionality, such as the affect engine *ALMA* (Gebhard, 2007) in *VirtualHuman* that computes changes in the emotional state of virtual characters, or a *narration engine* to provide external goals for the virtual characters. The framework is intended to run in a dialogue system environment where separate concurrent and specialized modules communicate using asynchronous message-passing. It implements a system of concurrent agents where each agent represents a conversation participant in a dedicated CDE sub-module. Besides an action manager, a CDE can host additional character-specific software instances, such as modules dealing with dialogue history or reference resolution. Indeed, in multimodal setups, the conversation manager is dependent on cooperating with an external module that provides for multimodal fusion and reference resolution; in all system instances described in this thesis, this task was taken over by the *FADE* module (Pfleger, 2007).

6.2 CDE Framework Architecture

6.2.1 Overview

Figure 6.1 shows how the components of the conversation manager framework are related to each other for a system instantiation with multiple characters and human users (the *Virtual-Human* system).

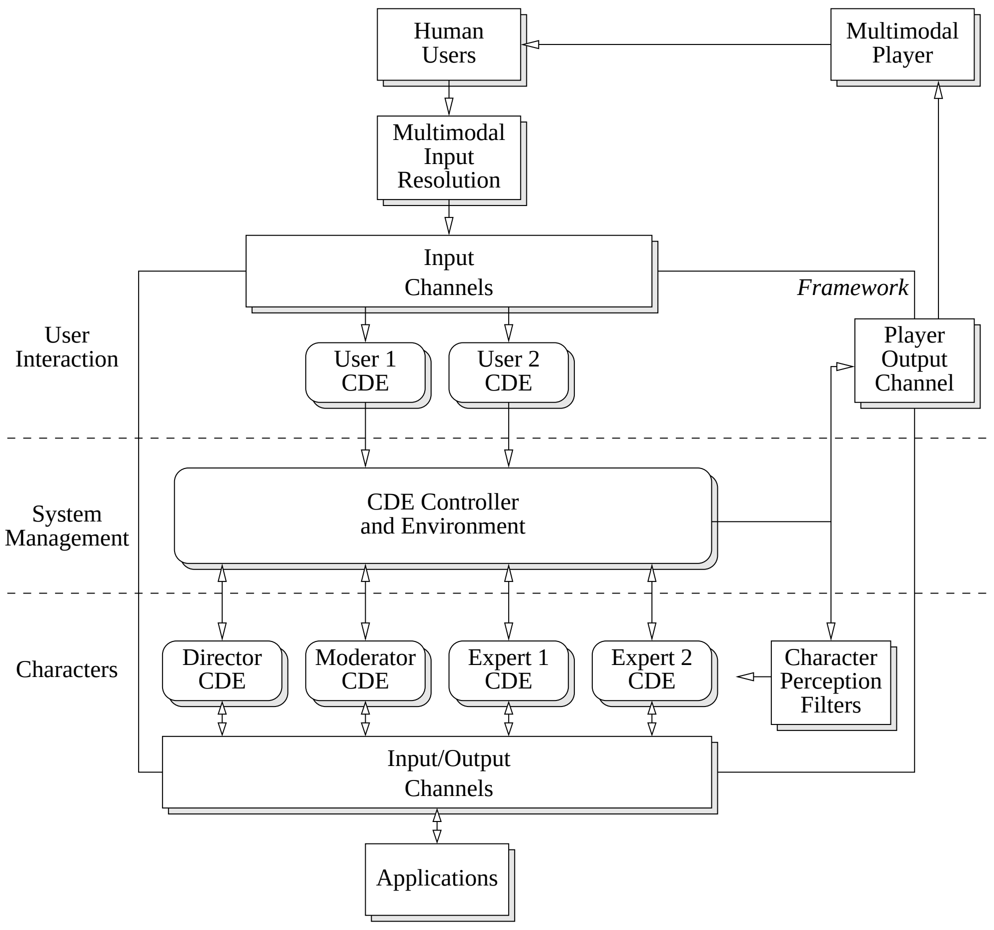


Figure 6.1: Overview of the internal structure of an exemplary conversation manager instantiation (*VirtualHuman*) with several human users and characters

There are three main areas, from top to bottom: On top are the parts concerned with input from and output to the human users, the middle part deals with the dialogue system control and managing the virtual environment, and the bottom part generates the behavior of the virtual characters, including possible communications with application modules (the system and character ontologies are omitted for simplicity).

The heart of the framework is the central *CDE controller*. It is initialized with a definition of a specific dialogue system and hosts the multi-agent system. It connects to the other modules of the system to receive and send messages and dynamically manages the CDEs, and also holds the system's ontology that represents the objective state of the environment. The controller is able to dynamically create new CDEs, or to deactivate present CDEs. This is done during the initialization of the system, according to its configuration, or dynamically during the interaction according to directives from a narration module.

6.2.2 Accessing the Knowledge Representation: *JenaLite*

The framework needs a means to access and manipulate the system ontology and the ontologies of the characters. A commonly used tool for this task in Java is the open-source *Jena* library.[1] However, while the *Jena* API and corresponding implementation are comprehensive and powerful, after initial experiments it showed that the time and space overhead imposed by the full implementation of RDF(S) was too grave, so that the real-time requirements of our system would not have been satisfiable if we used *Jena*.

To address this problem, a lighter API, *JenaLite*, was specified and implemented that operates on the TFS ontology representation generated from the RDFS knowledge base, as described in section 4.3.4 and is restricted to the features that are actually needed by the framework. The API then was implemented in two versions in a student project, yielding the ability to handle ontologies stored in the XML format, and in RDF(S). Internally, the implementation uses the speed-optimized DOM library JDOM to hold and manipulate the ontological taxonomy and the instances.[2] The use of *JenaLite* resulted in considerable speed and memory savings. Appendix B lists the elements of the *JenaLite* API.

6.2.3 Configuration

6.2.3.1 Dialogue System Definition

The dialogue system is specified by the set of interrelated knowledge sources shown in figure 6.2. The main resource is the *Dialogue System Definition* (DSD) that configures the controller for the dialogue system (the schema definition for DSD documents can be found in Appendix C.1). An initial *controller section* specifies the system ontology, the communication channel definitions for connecting the system to the outside world, and the initial actions that should be executed at the initialization of the system. The controller section is followed by the *participants section*, in which the CDEs that occur in the system, representing virtual characters or users, are defined. The DSD can also optionally contain an *application stub* to start external applications.[3]

[1]*Jena* website: *http://jena.sourceforge.net*
[2]*JDOM* website: *http://www.jdom.org*
[3]In case the conversation manager is to start other modules on initialization, this item is used to specify a stub Java class that handles their initialization.

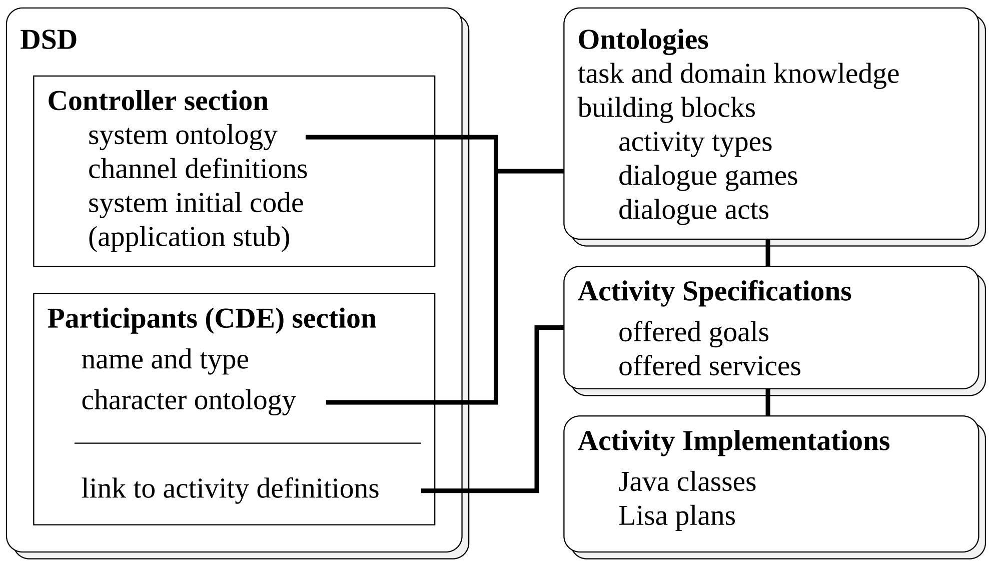


Figure 6.2: The set of knowledge resources for a CDE system

Figure 6.3 shows the dialogue system definition for the *OMDIP* system. In the controller section, the path to the system ontology is given. *OMDIP* uses three communication channels, to the *FADE* module (Pfleger, 2007), the GUI generator, and to the function modeler that interfaces to the external application services. The *type* of a channel gives a Java class that is dynamically loaded to implement the translation function. Since in the case of *OMDIP*, all communication is in the same common data format (*EMMA*), a single class *OmdipChannel* can be used for all channels. There is no application stub given. Finally, the controller section can optionally contain *initial code* in the *Lisa* language (see Section 6.4) that is executed when the system is initialized. In this case, two CDEs are created and activated at the beginning.

The *participants* section specifies that the system can use two CDE instances, one for the user and one for the system agent. Again, CDE types are supplied which identify Java classes that are subclasses of an abstract *CDE* class. Both CDEs and the CDE controller use the same ontology, which is contained in an XML file that contains the ontology translated to the XML representation described in Section 4.3.4. For the system CDE, an activity definition file is specified; the *OmdipUserCDE* does not require activity definitions, since it is only a proxy for the user.

The DSD enables the controller to initialize the conversation manager in four steps:

1. Create the initial state of the system environment by loading a file containing the system ontology,

2. Prepare all CDE instances that are in the DSD, and initialize them with their activity specifications,

3. Set up the input and output channels for external communication, and

4. Execute additional arbitrary instructions necessary to bring the system to its initial state.

```
dsd
  controller
    name: OMDIP
    ontology: etc/ontology/omdipOntology.xml
    subdir: omdip
    gameTypesPath: etc/dialog/gameTypesOMDIP.xml

    channels
      channel
        name: fade
        type: de.dfki.omdip.OmdipChannel
      channel
        name: fm
        type: de.dfki.omdip.OmdipChannel
      channel
        name: GUIGen
        type: de.dfki.omdip.OmdipChannel

  participants
    cde
      name: OMDIPAgent
      type: de.dfki.omdip.OmdipCDE
      ontology: etc/ontology/omdipOntology.xml
      activityDefinitionFile: etc/omdip/omdipAgent/omdipAgent.xml

    cde
      name: OMDIPUser
      type: de.dfki.omdip.OmdipUserCDE
      ontology: etc/ontology/omdipOntology.xml

  init
    CreateCharacter
      has_character
        Character
          has_name: OMDIPAgent
    CreateCharacter
      has_character
        Character
          has_name: OMDIPUser
```

Figure 6.3: The DSD file for *OMDIP*

This can involve sending further initializations to other modules in the system, activating CDEs, or setting initial goals for character CDEs.

After the initialization, the system is ready to engage in conversation and can process user input or directions from a narration engine.

6.2.3.2 Activity Specifications

The activity definition file contains the definitions for all activity types and the services they offer. Figure 6.4 shows an excerpt of three activity type definitions from the activity definition file for the *OMDIP* system agent.

As shown in the figure, an activity type definition specifies the following information:

- the *name* of the concept in the ontology representing the activity type. This concept will typically contain roles that can hold object instances relevant to the activity, acting as local variables.
- the *type* of the activity executable, and the implementing Java *className*. There are three types in the standard framework, *LisaProcess*, *HardcodedProcess* and *ManagedProcess* (see Section 6.3.3).
- a set of *service* templates for the activity type.
- whether the services of the activity type should be automatically registered (*autoRegister*) and/or started when the CDE is initialized (*autoStart*).

The set of service templates are interpreted as underspecified TFS instances that can be registered in the *registeredServices* table of the CDE, which maps from templates to activity types. In the third activity type in the example, this would result in the following mappings being included in *registeredServices*:

$$\begin{bmatrix} \textit{Command} & \\ \textsc{has\_content} & \begin{bmatrix} \textit{Offering} & \\ \textsc{has\_object} & \begin{bmatrix} \textit{Service} & \\ \textsc{has\_name} & \textit{ServiceCenter} \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad \longrightarrow \quad \textit{ServiceCenterProcess}$$

$$\begin{bmatrix} \textit{Select} & \\ \textsc{has\_content} & \begin{bmatrix} \textit{Service} & \\ \textsc{has\_name} & \textit{ServiceCenter} \end{bmatrix} \end{bmatrix} \quad \longrightarrow \quad \textit{ServiceCenterProcess}$$

Registered services can be triggered by user interaction or *SetGoal* acts by external modules. If services are not registered automatically, they can still be called as sub-activities from other activities.

```
processes
  ActivityProcessType
    has_type: lisa
    has_name: NotAvailableProcess
    has_className: de.dfki.cde.process.LisaProcess
    has_autoRegister: true
    has_autoStart: false
    has_service
      DialogueAct
    has_plan: etc/omdip/omdipAgent/notAvailable.xml
  ActivityProcessType
    has_type: lisa
    has_name: ComeAgainProcess
    has_className: de.dfki.cde.process.LisaProcess
    has_autoRegister: true
    has_autoStart: false
    has_service
      NotUnderstood
    has_plan: etc/omdip/omdipAgent/comeAgain.xml
  ActivityProcessType
    has_type: lisa
    has_name: ServiceCenterProcess
    has_className: de.dfki.cde.process.LisaProcess
    has_autoRegister: true
    has_autoStart: false
    has_service
      Select
        has_content
          multimod_Service
            has_name: ServiceCenter
    has_service
      Command
        has_content
          smartsumo_Offering
            multimod_has_object
              multimod_Service
                has_name: ServiceCenter
    has_plan: etc/omdip/omdipAgent/serviceCenter.xml

    [ ... ]
```

Figure 6.4: Excerpt from the activity definition file for the OMDIP system agent

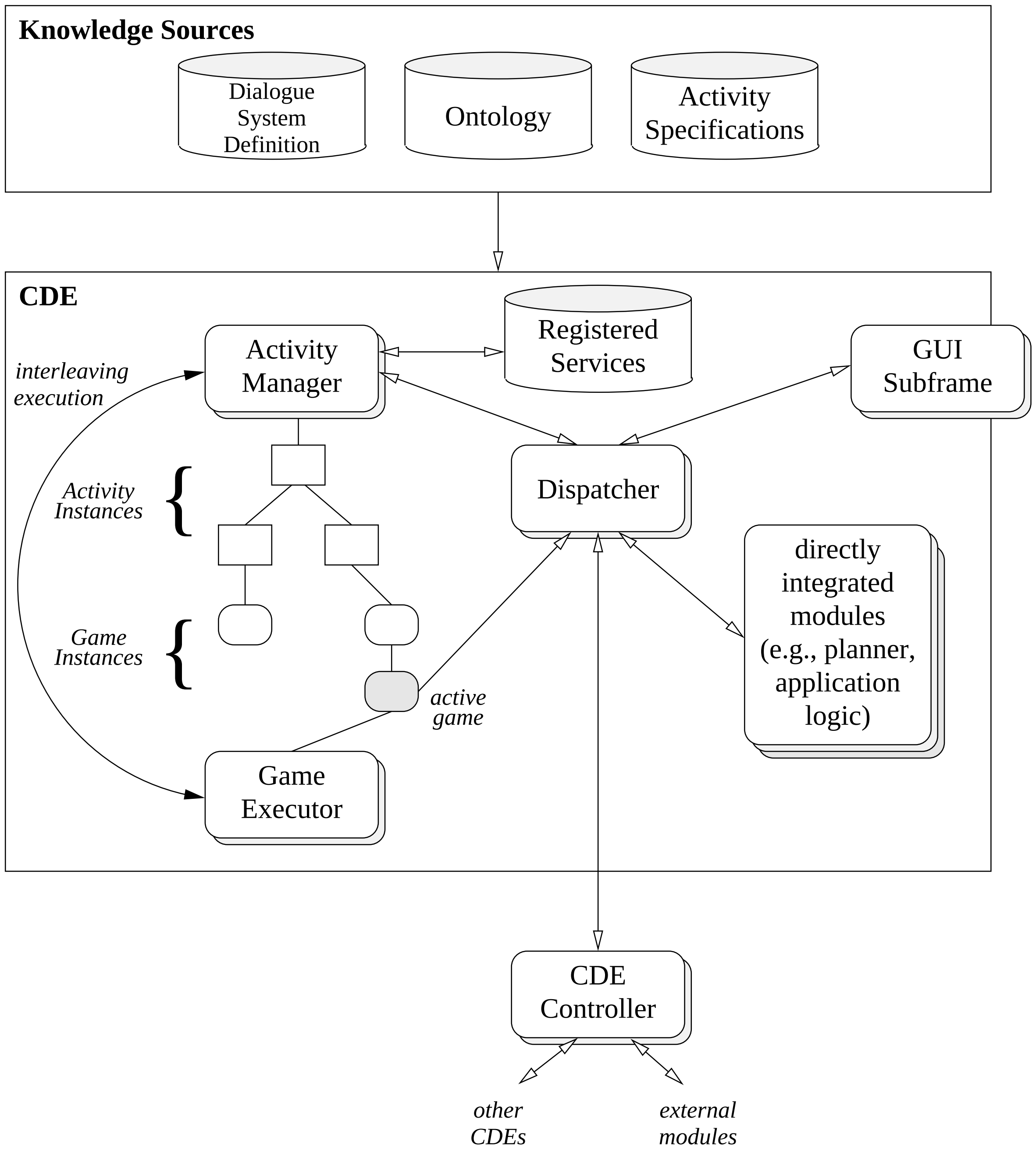


Figure 6.5: The essential internal composition of a CDE (without system-specific extensions)

6.2.4 CDE internal structure

Without any configured modules and channels, a plain CDE has an internal processing structure as shown in figure 6.5. This basic structure is designed to be modified by subclassing the *CDE* class from the framework. The CDE draws on the knowledge sources DSD, ontology and the activity specifications from the configuration. A *dispatcher* distributes incoming and outgoing messages.

The *activity manager* controls the hierarchy of currently active and suspended activity processes, and schedules their execution. It is parameterized by the registered services, which can be either static (from the activity specification) or dynamically registered by other running activities. Each activity is running as a separate thread in the engine. The activity manager executes in an alternating fashion with the *game executor* that becomes active when a game has to be advanced by generating acts of the character or by consuming user acts.

Additional modules can be integrated directly into subclasses of the CDE class, instead of connecting them via channels in the CDE controller for improved performance (this was done, e. g., in *VirtualHuman* for the integration of *FADE* and the module implementing the game logic for the lineup game). Each CDE also provides a GUI subframe that is integrated into the overall framework GUI managed by the CDE controller.

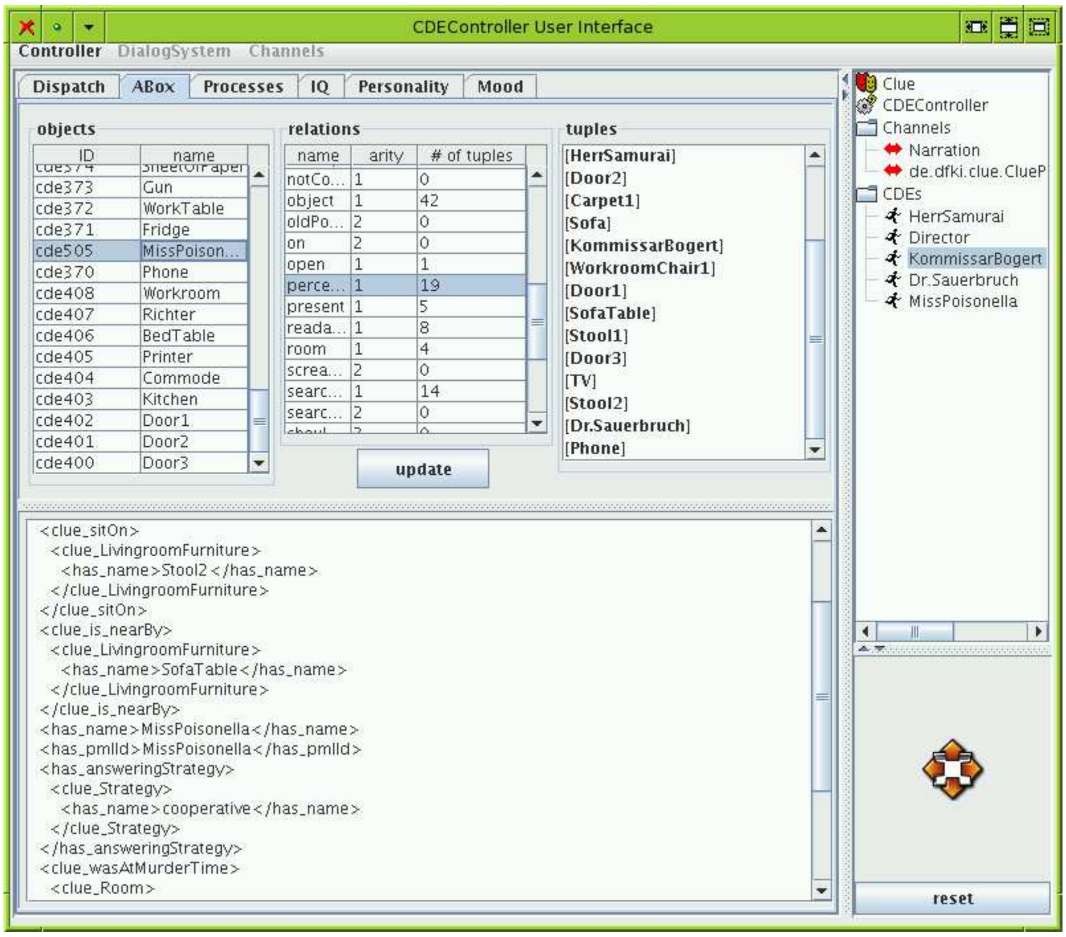
6.2.5 The Framework's Graphical Interface

A graphical user interface allows the conversation designer to examine and influence many aspects of the system in real-time. This can be especially useful for debugging, testing, and monitoring a live system during development, and also makes it possible to configure specific situations interactively.

The main features of the interface are:

- **Character CDEs**
  - **Ontology View**
    This view allows the dialogue designer to browse through and inspect the instances in the ontology of an active character and step-wise track changes in the relations between them (see Figure 6.6).
  - **Act Dispatch**
    Using this feature, dialogue acts can be directly input or selected from a list of files to be sent as if the behavioral module of the character generated them. This is especially useful when the system exhibits partially non-deterministic behavior for the sake of variation in a narrative, e. g., characters selecting randomly from a set of possible actions.

    In this case, some possible interactions may show up infrequently, making them hard to test. This difficulty can be alleviated by manual testing. Additionally, the effect of new behaviors can also be tried out manually before adding them to the knowledge base proper.



ID	name
cde373	Gun
cde372	WorkTable
cde371	Fridge
cde505	MissPoison...
cde370	Phone
cde408	Workroom
cde407	Richter
cde406	BedTable
cde405	Printer
cde404	Commode
cde403	Kitchen
cde402	Door1
cde401	Door2
cde400	Door3

name	arity	# of tuples
notCo...	1	0
object	1	42
oldPo...	2	0
on	2	0
open	1	1
perce...	1	19
present	1	5
reada...	1	8
room	1	4
screa...	2	0
searc...	1	14
searc...	2	0

[HerrSamurai]
[Door2]
[Carpet1]
[Sofa]
[KommissarBogert]
[WorkroomChair1]
[Door1]
[SofaTable]
[Stool1]
[Door3]
[TV]
[Stool2]
[Dr.Sauerbruch]
[Phone]

```
<clue_sitOn>
  <clue_LivingroomFurniture>
    <has_name>Stool2</has_name>
  </clue_LivingroomFurniture>
</clue_sitOn>
<clue_is_nearBy>
  <clue_LivingroomFurniture>
    <has_name>SofaTable</has_name>
  </clue_LivingroomFurniture>
</clue_is_nearBy>
<has_name>MissPoisonella</has_name>
<has_pmlId>MissPoisonella</has_pmlId>
<has_answeringStrategy>
  <clue_Strategy>
    <has_name>cooperative</has_name>
  </clue_Strategy>
</has_answeringStrategy>
<clue_wasAtMurderTime>
  <clue_Room>
```

Figure 6.6: Ontology view for officer Bogert in the *Clue* system

– **Processes**

  This view allows an operator to supervise the active goals of the character and their completion status.

– **IQ, Personality and Mood**

  For these, a set of views is provided that dynamically display the configuration of the character in terms of traits and affective parameters.

• **Channels**

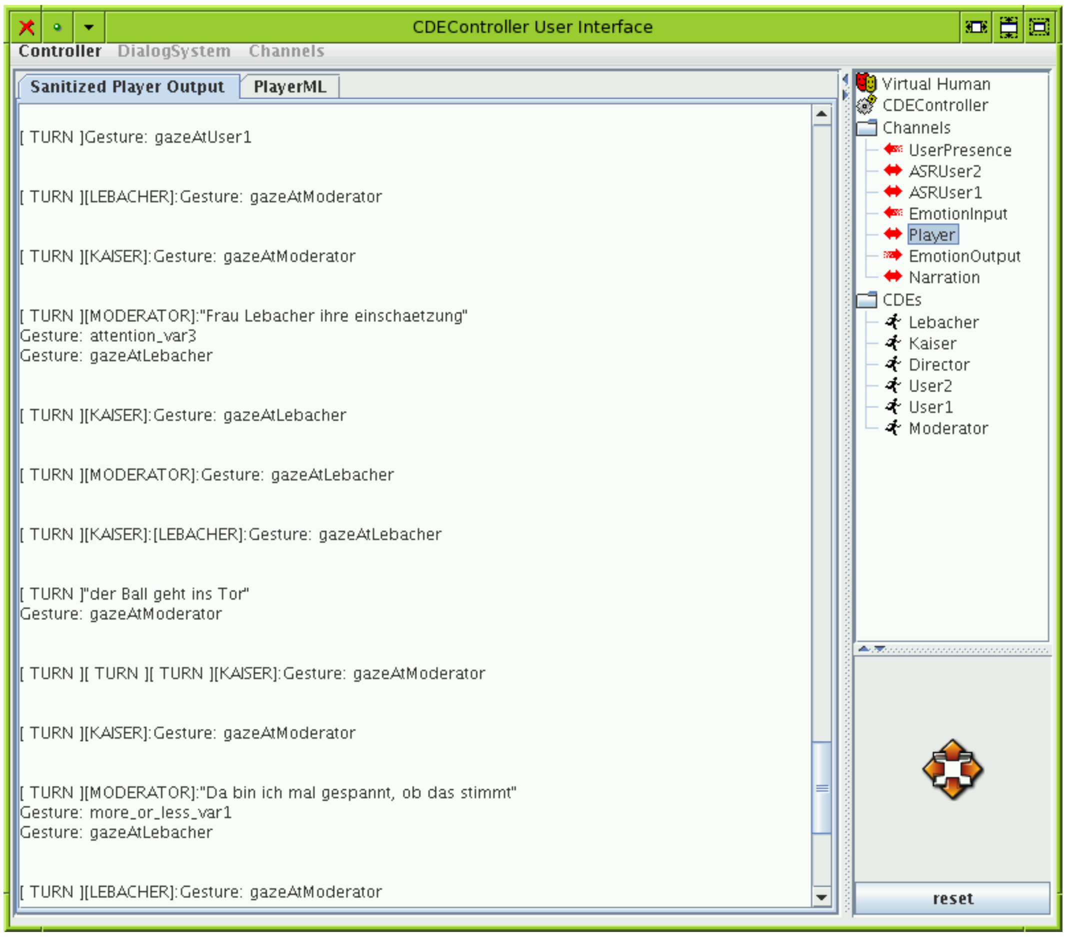
The channel views are dependent on the application configuration. The following possibilities are available if the corresponding channel is present:

– **Speech Input**

  The operator can use the keyboard to provide a speech recognition result for a human user's CDE, and let it be processed by the system.

– **Affective Input and Output**

The input view shows the updates by the affect engine in real-time, while the output view protocols the affective tags that are sent by the CDEs to the affect engine.



```
[ TURN ]Gesture: gazeAtUser1

[ TURN ][LEBACHER]:Gesture: gazeAtModerator

[ TURN ][KAISER]:Gesture: gazeAtModerator

[ TURN ][MODERATOR]:"Frau Lebacher ihre einschaetzung"
Gesture: attention_var3
Gesture: gazeAtLebacher

[ TURN ][KAISER]:Gesture: gazeAtLebacher

[ TURN ][MODERATOR]:Gesture: gazeAtLebacher

[ TURN ][KAISER]:[LEBACHER]:Gesture: gazeAtLebacher

[ TURN ]"der Ball geht ins Tor"
Gesture: gazeAtModerator

[ TURN ][ TURN ][ TURN ][KAISER]:Gesture: gazeAtModerator

[ TURN ][KAISER]:Gesture: gazeAtModerator

[ TURN ][MODERATOR]:"Da bin ich mal gespannt, ob das stimmt"
Gesture: more_or_less_var1
Gesture: gazeAtLebacher

[ TURN ][LEBACHER]:Gesture: gazeAtModerator
```

Figure 6.7: The player preview showing a conversation between moderator (Kaiser) and expert (Lebacher) in *VirtualHuman*

- **Presentation Output and Feedback**

  This view shows for all characters the generated utterances, including gestures, physical acts, and player configuration messages. The output is available in terms of the actual message sent to the 3D player in *playerML* (player markup language), and a more human-readable "sanitized" format, as shown in Figure 6.7.[4]

- **Narration Input and Output**

  In this view, an operator can send single narrative goals manually and examine the status all current narrative goals, as well as *SetGoal* and *GoalFeedback* messages. Section 7.2.2 features a screenshot and a more detailed description of its possibilities.

[4]A description of PML can be found in (Jung and Knöpfle, 2007; Klesen and Gebhard, 2007)

- **Reset**

  The conversation can be restarted from the GUI. If other modules in the system support remote restart, they are also sent reset messages.

6.2.6 Connecting Modules and Devices to the Framework

The system is designed to enable easy interoperation with off-the-shelf modules and devices that are not part of the framework. While the content of messages generated by modules of the framework is based on the common ontological representation, communication from and to external programs (e. g., speech recognizers and multimodal players) has to be adapted to this scheme. Usually, changing the source code of such programs is not an option. To avoid having to integrate idiosyncratic conversions into the core of CDE framework itself, it provides a set of interfaces that define the functionality required for interaction with external applications.

For the integration, it is necessary to provide Java classes that implement the interfaces *InputChannel* (for incoming messages) and *OutputChannel* (for outgoing messages) and realize the conversion function as defined in Section 5.3.3. The qualified names (including the package specification) of these classes, and a name for the associated channel, can then be specified in the DSD for the dialogue system. The system will load and instantiate them dynamically at run-time. CDEs that need to be interfaced to external modules or devices can subscribe to the respective input channel or publish messages for an output channel. Instantiated channels also receive a message when the system is reset, in case the modules or devices need to be notified of this event.

6.3 Performing Activities and Dialogue Games

Section 5.5.3 stated that the behavioral and deliberative actions of the characters are performed in three modes of action: deliberation, consuming acts, and initiating acts. Here, we describe how these action modes are managed for each CDE. We start with how goals are set and how goal feedback is generated, introduce the notion of expectations, and then turn to the implementation of activities.

6.3.1 Goals

6.3.1.1 Setting Goals

There are four cases for how an activity can be triggered for a character:

(1) automatically at the initialization of the system,

(2) implicitly to process a dialogue act that no current activity can process.

(3) explicitly by another activity,

(4) explicitly by an external source, e. g., a narration engine,

Case (1) is carried out while loading the DSD (see Section 6.2.3.1). (2) is handled by the act consuming algorithm (see Section 5.7.6.1). The cases (3) and (4) involve passing a *SetGoal* meta-act to the character. An activity can do this either via a command in the *Lisa* language (section 6.4) or an explicit call to the framework API; an external module has to send a message via an input channel.

An activity continues to execute until it terminates. If it was started by an external source, a feedback message is sent to the goal initiator upon termination that indicates the termination states "success" or "failure" and possibly additional information. The basic type of *SetGoal* contains the roles shown in Figure 6.8.

$$\begin{bmatrix} \mathit{SetGoal} & \\ \text{HAS\_NAME} & \mathit{String} \\ \text{HAS\_INITIATOR} & \mathit{Agent} \\ \text{HAS\_ADDRESSEE} & \langle \mathit{Character} \rangle \\ \text{HAS\_TIMEOUT} & \mathit{Integer} \\ \text{HAS\_EVENT} & \langle \mathit{Event} \rangle \\ \text{HAS\_ACTIVITY} & \mathit{Activity} \\ \text{HAS\_BEGINTIME} & \mathit{Time} \\ \text{HAS\_ENDTIME} & \mathit{Time} \end{bmatrix}$$

Figure 6.8: Roles of the *SetGoal* concept

The external narration engines in *VirtualHuman* and *Clue* use the *directionML* markup language for sending goals and receiving feedback (the schema definition of *directionML* messages is given in appendix C.2). A directionML message setting the *Lineup* goal of *VirtualHuman* is shown in figure 6.9.

Upon receiving this message, the CDE controller will construct and send *SetGoal* acts to all participants specified in the message (the user CDE ignores goals). Under the *Goal* tag, an ontological instance is specified in XML notation, which takes on the *activity* role in the *SetGoal* act. An activity can take a *SetGoal* message directly to the CDE it runs in, but not to other CDEs.

6.3.1.2 Goal Feedback

When goals are imposed on characters by an external entity such as a narration engine, it can be crucial to have a possibility to give back notifications about the state of the activities. This is done by sending *goal feedback messages* to the goal initiator upon the following events:

- *Termination:* When a goal terminates successfully or unsuccessfully.
- *Timeout:* After the duration of *timeout*, the activity automatically terminates with failure. In the example, this would be after 360 seconds.
- *NoResponseEvent:* When the activity associated with the goal does not send or receive any communicative acts for the duration specified in the *SetGoal* timeout (after 60 seconds in the example).

```
directionML
  setGoal
    has_name: lineup
    Participant
      has_name: Moderator
    Participant
      has_name: Herzog
    Participant
      has_name: User1
    Timeout: 360 // timeout in seconds
    NoResponseEvent: 60
    Event refId="lastEvaluation"
    Goal
      zamb_Lineup
        has_moderator
          Character
            has_name: Moderator
        has_expert
          Character
            has_name: Herzog
        has_contestant
          Character
            has_name: User1
        has_opponent
          FootballTeam
            has_name: Japan
        has_lastEvaluation
          Evaluation id="lastEvaluation"
```

Figure 6.9: A *directionML* message setting the *Lineup* goal

- *Events:* When activity roles are marked by reference in an *Event* instruction change (the value of the *has_lastEvaluation* role in the example). In this case, the new value of the role is sent back in the feedback as a *result*

$$\begin{bmatrix} \textit{GoalFeedback} & \\ \textsc{has\_name} & \textit{String} \\ \textsc{has\_status} & \textit{String} \\ \textsc{has\_initiator} & \textit{Character} \\ \textsc{has\_event}^{*} & \textit{Event} \\ \textsc{has\_result} & \textit{:THING} \end{bmatrix}$$

Figure 6.10: Roles of the *GoalFeedback* concept

A *GoalFeedback* meta-act has the roles shown in Figure 6.10, which encode the aforementioned information.

Figure 6.11 shows an abridged possible feedback message for a termination event. It was sent by the “Moderator” character in response to the goal set by the message in Figure 6.9.

```
directionML
  goalFeedback
    has_name: lineup
    has_initiator
      Character
        has_name:  Moderator
    has_status: failure
    has_event
      Timeout
    has_result
      ZambEvaluation
        has_evaluatedObject
          ZambMove ...
        has_globalScore
          Score
            has_character
              Character
                has_name: User1
            has_points: 67
        ...
```

Figure 6.11: An example *GoalFeedback* message

It indicates that the Lineup goal failed because of a timeout event, and that the player had a score of 67 points when the activity was aborted.

6.3.2 Expectations

Dialogue processing algorithms can exploit the fact that it is often possible to predict future utterances. Sources for such predictions are the social obligation expressed in the dialogue games, but also from the wider task or story context of the interaction. The benefits of using expectations to place constraints on input disambiguation has already been exploited in various systems, e. g., in *RavenClaw* (Bohus and Rudnicky, 2003) and in form of anticipation feedback loops in the *PRACMA* system (Ndiaye and Jameson, 1996). Usually, expectations are considered binary, i. e., an utterance is either expected or not. Our approach uses a somewhat more fine-grained distinction between several degrees of expectancy. These expectation categories were introduced in (Löckelt et al., 2002) for the *SmartKom* system. There, the expectations related to expected *slots*, i. e., slots that were expected to be filled, but the principle can be transferred. In the model described in this thesis, they refer to expected *dialogue acts* by other conversation participants.

- An *expected act* in a dialogue game is an act associated with an edge outgoing from the current state it is in. The initiative label of the edge also must specify the interaction counterpart, i. e., if the expecting party (*"expectator"* is the initiator of the game, the edge must be labeled with the initiative value "responder", and vice versa.

- *possible acts* are all acts that start a dialogue game in currently executing activities

- *other acts* are acts that cannot be reasonably interpreted in the current context, for example acts that would start a new activity.

Expectations play a central role in routing incoming utterances for a character to the correct activity. If a dialogue game explicitly "expects" an act, it will be given preference over an activity that only declares it "possible", and only if the act does not fall into either category, does a new activity have to be started. Section 5.7.6.1 states how an expected act in a dialogue game is determined by examining the outgoing edges of its current state. The expectation information is sent via an output channel in the form of an *Expectation* data structure to other modules on the input analysis side, especially *FADE*, to help with disambiguation (cf. (Löckelt et al., 2002)).

In addition to the status of the various slots of the different processes, the expectation data structure can also contain *lexicon updates*. If the system features a dynamic lexicon, it can subscribe to the expectation messages and receive new sets of words that are likely to be used in the current application context. Consecutive lexicon updates are collected and dispatched together the next time an expectation is sent. Below are two sample statements in the *Lisa* language to construct a lexicon update after new song titles and artist names are retrieved from the database in *OMDIP*:

```
lexiconUpdate
  slotName: performanceList
  path: listItem/SoundLogo/has_title
  category: songTitle
lexiconUpdate
  slotName: performanceList
  path: listItem/SoundLogo/has_artist/Artist/has_name
  category: artist
```

Each of these statements will collect the (string-valued) objects found under the given *path* in the role with the name *slotName* and construct a lexicon update with these objects as lexemes for the category *category*.[5] Figure 6.12 shows an example of an *Expectation* instance containing two lexicon updates that could result from this expression.

6.3.3 Activities

All activities in a CDE are created in a process hierarchy as children of a *root activity* (cf. Figure 5.5 on page 132).

They are in one of the different states shown in Figure 6.13 at all times. Activities themselves can spawn other activities, either as root activity children or as sub-activities. In the latter case, the original activity is in suspended state until the child activity terminates. The main activity manager loop executes all non-suspended activities in turn (Figure 6.14). If there is an incoming act in the activity's input queue, it has to process it via a *consume()* method, otherwise an *execute()* method is called that invokes the deliberation mode, from where the

[5]In *Lisa*, references to substructures of ontological instances are given in XPath syntax to be applied to their XML representation.

$$\left[\begin{array}{ll} \textit{Expectation} & \\ \textsc{is\_expected}^{*} & \textit{Act} \\ \textsc{is\_possible}^{*} & \textit{Act} \\ \textsc{is\_other}^{*} & \textit{Act} \\ \textsc{has\_lexiconUpdate} & \left\langle \left[\begin{array}{ll} \textit{LexiconUpdate} & \\ \textsc{wordClass} & \textit{songTitle} \\ \textsc{objectType} & \left[\textit{SoundLogo}\right] \\ \textsc{lexeme} & \left\langle \begin{array}{l} \textit{Music,} \\ \textit{Mensch,} \\ \textit{Material Girl} \end{array} \right\rangle \end{array}\right], \left[\begin{array}{ll} \textit{LexiconUpdate} & \\ \textsc{wordClass} & \textit{artist} \\ \textsc{objectType} & \left[\textit{Person}\right] \\ \textsc{lexeme} & \left\langle \begin{array}{l} \textit{Madonna,} \\ \textit{Herbert Grönemeyer} \end{array} \right\rangle \end{array}\right] \right\rangle \end{array}\right]$$

Figure 6.12: An example *Expectation* containing a lexicon update

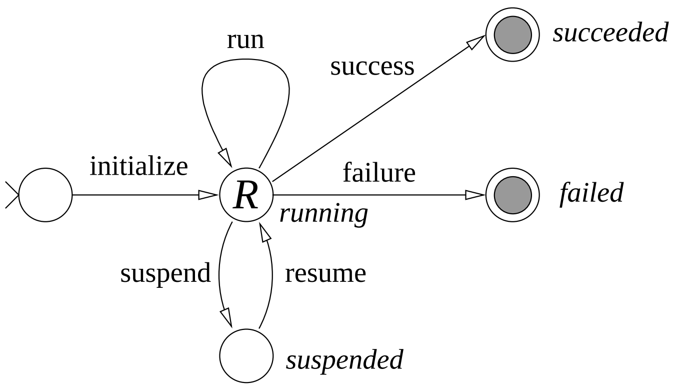


Figure 6.13: Process states for activities (adapted from (Hulstijn, 2000b))

activity can also initiate an act of its own. If the activity has reached the *finished* state afterwards, the *send-feedback* method will notify the initiator of the corresponding goal, if it is an external module.

To service the input queues of the activities, a separate *dispatcher* thread runs concurrently and is called whenever a new act is received from any channel. The dispatcher distributes the acts to the input queues of the activities managed by the process manager via the procedure shown in Figure 6.15 that determines the right processing level and the destination activity, as was illustrated in Figure 5.25.[6] The combination of the activity manager loop and the dispatcher ensure that an act is sent to the correct activity and processed when it is scheduled next for execution.

To implement the activity, several possibilities are available. The DSD specifies which implementation type is used for each activity by the *type* and *className* features. Figure 6.16 shows the basic Java classes available in the framework for executing activities. The basic implementations, except *LisaProcess*, are abstract classes and must be extended by subclassing before they can be used.

To act as an activity implementation, a class must implement three basic methods of the *Pro-*

[6]The additional case where an act has an explicit receiver process covers CDE-internal communication, such as an answer from a planner module.

```
ACTIVITY-MANAGER-LOOP ():

loop
  forall activity ∈ Activities where activity is not suspended
    if activity is not finished
      if there an incoming act act in activity's input queue
        activity.consume(act)
      else
        activity.execute()

    if activity is finished and goal originated from external source
      send−feedback(activity.getState())
```

Figure 6.14: The activity manager loop

cess class; these methods are needed for interaction with the activity manager. CONSUME(*Act*) is called to deliver an act to the activity, EXECUTE() lets the activity do internal computations, and GETSTATE() has to return the running state of the activity (*running*, *suspended*, *succeeded*, or *failed*). There are two other abstract subclasses directly inheriting from the basic *Process*. Subclassing *HardcodedProcess* is intended for free implementation for the processing of the acts in plain Java, in case some desired behavior is not capturable with the dialogue game paradigm. *ManagedProcess* offers some basic convenience methods for implementing dialogue game-based processing "manually". *LisaProcess* again extends *ManagedProcess* and can execute activities based on activity *plans* in the *Lisa* language, which we describe next.

6.4 The *Lisa* Language

Figure 6.17 summarizes the top-level algorithm for the execution of dialogue games (both single-initiative and multi-initiative) in pseudo-code, including deliberation, consuming acts, and realizing own acts. *Lisa* is a special-purpose language for parameterizing the basic algorithm.

6.4.1 Motivation

Each dialogue application has its own set of idiosyncratic requirements. A dialogue designer can find it difficult to realize required features in the absence of the versatility of a full general-purpose programming language. Examples that are encountered in real systems are, e. g.

- extraction of patterns from ontological data structures for lexicon updates using regular expressions (*SmartKom*)
- specification of rules that are not conversation related, e. g., the rules for placing a legal football team on a field (*VirtualHuman*) or for deciding on a strategy to solve a criminal case (*Clue*)

```
DISTRIBUTE (act):

 if there is an explicit receiver process for act
   add act to receiver's input queue
 else if there is a nonempty set expectators of dialogue games expecting act
   let $e \leftarrow$ the strict best match for act in expectators
   add act to $e$'s input queue
 else if there is a set of process types $s$ offering a matching service
   let $t \leftarrow$ the strict best match for act in $s$
   if an instance $i$ of type $t$ is running
     add act to $i$'s input queue
   else
     start a new process instance $i$ of type $t$
     add act to $i$'s input queue
 else
   if the root process can handle act
     add act to the root process' input queue
   else
     reject(act)
```

Figure 6.15: Distribution algorithm for the dispatcher

- adapting output modes to differing presentation paradigms, e. g., as a 3D scene (*VirtualHuman* and *Clue*) vs. a web presentation (*OMDIP*)

The most direct way to implement services is to explicitly write them in a programming language such as Java. However, this has several drawbacks, e. g., lack of abstraction, difficulties in enforcing compliance to the constraints of a dialogue model, poor integration with knowledge sources, and no possibility to support the dialogue designer with development tools.

On the other hand, it would be possible to use a general formalism such as description logic together with a general-purpose problem solver. Such an abstraction is quite removed from the actual communication situation. This approach, while powerful, tends to lead to overly verbose and obfuscating, and therefore also error-prone, descriptions with regard to the actual content. For this reason, it is cumbersome to implement complex applications using explicit logic representation. To employ the principle of adapting the programming language to the problem, instead of vice versa, also known under the name "bottom-up programming" (cf. (Graham, 1994)), various dialogue systems have used special-purpose languages such as HAP or ABL (see Chapter 3).

For the CDE framework, we implemented *Lisa* ("**L**anguage for **i**nteractions of **s**ituated **a**gents"). It is an XML-based language that can be used to declaratively specify the actions in the activities for the character CDEs. Its repertoire includes constructs to manipulate the belief state of the character, manage sub-processes and conduct communicative games, as well as procedural control constructs. *Lisa* is an imperative programming language designed to be extended incrementally. If new and useful patterns are identified by analyzing activity types that it cannot yet cover and that have to be hard-coded, they can be integrated into the

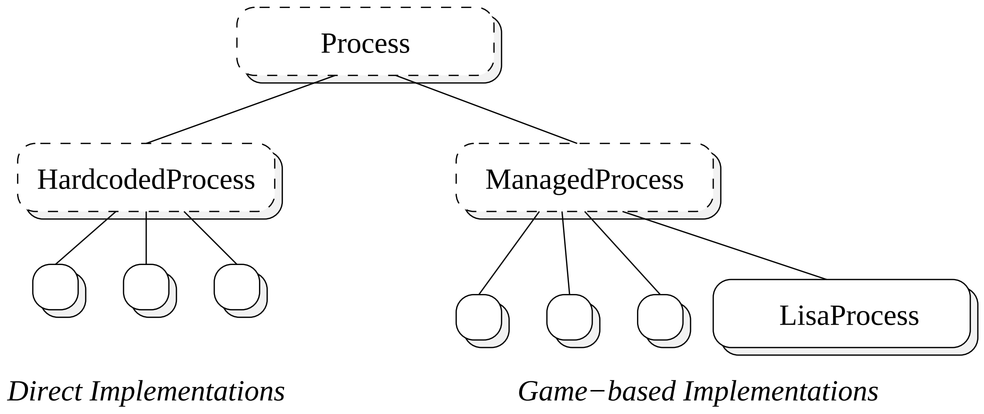


Figure 6.16: Activity implementation classes (classes in dashed boxes are abstract and must be subclassed)

language. The XML notation makes it possible to automatically verify *Lisa* specifications for syntactic correctness by using XSD checking (including ontological entities that are part of the specification) (Gurevych et al., 2006).

6.4.2 The Structure of a *Lisa* Plan

An activity specification in *Lisa*, also called a *Lisa* plan, declares the services offered by the activity and specifies how they are implemented. It can then be executed by an interpreter, which is provided by the *LisaProcess* class.

A *Lisa* plan consists of a *header*, *slot definitions*, an activity *body* and *dialogue game specifications*. It can also specify preconditions and postconditions for the activity. The header of the plan specifies the services it implements, and the name of the dialogue games that are used for the implementation. The slot definitions provide a way to store named ontological instances as local variables for the activity. The body of the activity is a sequence of instructions to be executed during the activity, and the dialogue game specifications provide instructions to be executed along with *DialogueGame* instances defined in the ontology.

As an example, we use a simple activity from the *OMDIP2* system called "ComeAgain". This activity offers a single dialogue game, called "ComeAgainGame", whose depiction as a graph is shown in Figure 6.18. The game is intended to be used to react to speech recognition errors. If such an error is detected, the input to the conversation manager consists of an instance of the concept *NotUnderstood* that may contain some semantic fragments, if the utterance could be partly interpreted by the input processing modules. The reaction of the system should consist in a dialogue act of type *ComeAgain*, whose *content* roles should contain (a) the semantic fragments, and (b) a counter indicating how many dialogue acts of the same type in sequence were not understood.[7] The basic "ComeAgainGame" instance is contained

[7]The rationale for this is that the presentation module might indicate a first, and only the first misunderstanding with a beeping signal, since several such signals in the case of consecutive misunderstandings would likely annoy the user.

```
DIALOGUEGAME.ADVANCE ():

 **let** *currentState* $\leftarrow$ state of game
 **case**: body of *currentState* has not been executed
   // deliberation
   execute body of *currentState*
 **case**: $g$ is final state
   // finishing game
   unregister expectations of $g$
   set game state to *finished*
 **case**: input queue contains an act $a$
   // consuming acts
   *applicableTransitions* $\leftarrow$ set of applicable transitions for $a$ from *currentState*
   *bestTransition* $\leftarrow$ strict best match for $a$ in *applicableTransitions*
   consume(*bestTransition, a*) // sets new currentState
   advanceGame(*bestTransition)*)
   unregister expectations of $g$
 **otherwise**:
   *applicableTransitions* $\leftarrow$ set of applicable edges for *currentState*
   **if** *applicableTransitions* $= \varepsilon$ and no expectations
   registered
     // expecting moves from other participants
     register expectations for new state
     publish expectations for other modules
   **else**
     // producing own move
     unregister expectations for $g$
     *bestTransition* $\leftarrow$ strict best match for $a$ in *applicableTransitions*
     realize(*bestTransition*)
     set current state of $g$ to target of *bestTransition*
```

Figure 6.17: Basic algorithm for a dialogue game's execution

in the ontology of the character. The additional *Lisa* plan uses this instance and specifies the additional operations to be performed during the execution of the game.

In the example of Figure 6.19, the service *comeAgain* is defined that is implemented by a game specification called *comeAgainSpec* that will follow below in the plan. Afterwards, three *slots* are defined to act as local variables for the activity, and given initial value assignments: *comeAgainContent*, *notUnderstoodCounter* and *lastNotUnderstood*. The body of the plan, in this case, contains a *loop* statement without a termination condition, i. e., an endless loop. That means that after the activity is started, it will not terminate until the whole application ends.[8]

The second part of the plan, shown in Figure 6.20, contains the game specifiation. It defines actions to be executed when the activity is in the different states of the game as a series of

[8]An empty endless loop in *Lisa* sleeps between iterations and does not take up much processor time.

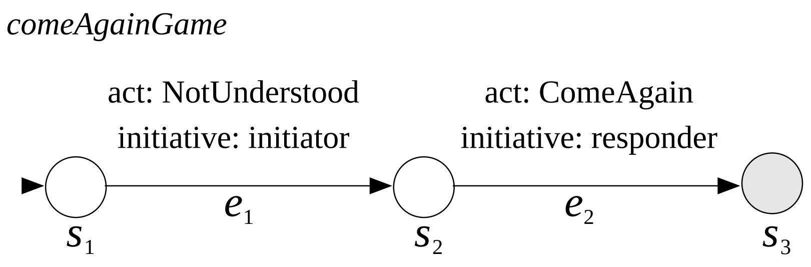


Figure 6.18: The *ComeAgain* game from *OMDIP2*

```
plan
  activity
    <!-- header -->
    name: comeAgain
    services
      service
        name: comeAgain
        usedGame: comeAgainSpec
        template
          NotUnderstood
      [...]
    <!-- slot declarations -->
    slots
      slot
        name: comeAgainContent
        value
          ComeAgain
      slot
        name: notUnderstoodCounter
        getABoxObject
          name: notUnderstoodCounter
      slot
        name: lastNotUnderstood
        value
          DialogAct
      slot
        name: counter
        string: 1
    <!-- activity conditions -->
    preconditions()
    postconditions()
    <-- activity body -->
    body
      loop()
```

Figure 6.19: Example of a *Lisa* activity specification, Part 1: header with service declaration, slot declaration, conditions, and main body (activity *ComeAgain* from *OMDIP2*)

```
games
  game
    name: comeAgainSpec
    gameType: comeAgainGame
    states
      state
        name: s2
        body
          setSlot(slotName: lastNotUnderstood,
                  getABoxObject(name: lastNotUnderstood)
          select
            template
              conditions
                condition
                  slotEqualsSlot(slotName: lastNotUnderstood,
                                 slotName: consumedAct)
              body() <!-- same as last time, do nothing -->
            template
              conditions() <!-- reset counter -->
              body
                replaceSlotAtPath(pathToSlot:notUnderstoodCounter:has_name,
                                  slotValue: counter)
          assertObject(name: lastNotUnderstood, slotValue: consumedAct)
          select
            template
              conditions
                condition
                  slotIsSet(slotName: content)
              body
                addSlotAtPath(pathToSlot: comeAgainContent:has_content,
                              slotValue: content
            template
              conditions()
              body()
          addSlotAtPath(pathToSlot: comeAgainContent:has_count,
                        getABoxObject(name: notUnderstoodCounter:has_name))
          setParameter(name: addressee, string: GUIGen)
          setParameter(name: content, slotValue: comeAgainContent)
          increment(slotName: counter)
          replaceSlotAtPath (pathToSlot: notUnderstoodCounter:has_name,
                             slotValue: counter)
[...]
```

Figure 6.20: Example of a *Lisa* activity specification, Part 2: dialogue game specification (activity *ComeAgain* from *OMDIP2*)

commands under the *body* tag. Only a body for state s_2 is given, since no other states require action. The operational semantics of the *Lisa* language constructs are given in Appendix A. We will not explain s_2's body in detail, but mention that the *getABoxObject* and *assertObject* statements retrieve resp. store object instances in the ontology under a given name. The overall effect of the procedural body can be stated as: "if the consumed act is different from the one stored in *lastNotUnderstood*, reset the counter. Store the consumed act in *lastNotUnderstood*. If there are semantic fragments in the consumed act, copy them to the *comeAgainContent* slot. Finally, set the parameter *content* for the next dialogue act to the value of *comeAgainContent* and increment the counter".

6.4.3 The *Lisa* Interpreter

The *Lisa* plans, which are XML documents, are executed by creating a *LisaProcess* object that uses an execution stack, a stack of bindings and an interpreter-parser for *body* code blocks (cf. Figure 6.20). Its mode of execution is similar to a SAX parser traversing a document. Therefore, the implementation makes use of a subclass of *SAXBuilder* (from the *JDOM* library) for the traversal. *Lisa* is an imperative language and features all necessary constructs (statements, values, assignments, recursion and control structures like conditional execution and loops). Via channel communication, it is possible to integrate external programs written in Java or any language (like a planner module).

When using the framework to build an application that involves a large number of concurrent communicative agents, it would be advantageous to alternatively compile *Lisa* programs to Java code snippets loaded at run-time to gain some speed improvement. Such an approach was described by, e. g., (Mateas, 2002) for the ABL language. However, with the low number of characters present in our scenarios (not more than four), the speed of the *SAXBuilder*-based implementation did not pose a problem.

It is also the case that the speed improvement attainable by a compilation to "native" Java code can only amount to a minor time constant per statement. *Lisa* is a high-level language; one *Lisa* statement can correspond to several dozen lines of java methods. For each statement that is executed by the interpreter, the performance loss from interpretation is only the time it takes to read the next node in an XML document and make a callback to a corresponding Java method in the *LisaProcess* class that implements the statement.

6.4.4 *Lisa* Language Elements

Lisa executable bodies feature three different types of language constructs (a full reference overview to *Lisa*'s language elements can be found in Appendix A)

- **Statements:**

  These comprise mainly commands manipulating slot contents and the ontology, like *assertABoxObject*, or the local slots, like *setSlot* or *replaceSlotAtPath*. Also, relations between objects can be created and manipulated by *createRelation* and *assert* statements. *Loop* constructs can be used to iterate over blocks of statements until a condition evaluates to false. *Select* statements allow conditional execution of code blocks. The *nextState* statement can be used to select one of several possible followup states of the game. A subgoal can be triggered by the *subgoal* statement that can be parameterized either to suspend the current goal until the subgoal has completed, or let both run concurrently. If the current goal suspends for a subgoal, it receives the local slot values of the subgoal at its completion in a data structure called *binding* that is named after the subgoal.

Several statements are put as children in a *body* statement to be executed sequentially.

- **Rvalues:**

  Rvalues are expressions that compute a value of type instance or String that can be assigned to slots or act as parameters for statements or conditions. They can use values as literals (*value*, *string*), from the local slot binding (*slotValue*), a binding from a subgoal (*bindingValue*) or the ontology (*getABoxObject*).

- **Conditions:**

  These are expressions that compute a boolean value for use in conditional statements such as *loop* and *select*. They can test for conditions between one or several slots (*slotHasValue*, *slotIsSet*, *slotEqualsSlot*) or whether a relation holds for a tuple of object instances in the ontology (*holds*).

When an activity specified by a *Lisa* plan is adopted as a goal, the CDE process manager starts a new *LisaProcess* that interprets the plan. It starts by registering the services that are not auto-registering with the process manager, initializes the plans' slot definitions, and then starts executing the main body. The process continues to run until its main body has been executed. If the activity is triggered by a service match, the main body is executed (if it is not already running), followed by the corresponding game specification.

When executing a game specification, transitions are made according to the algorithm in Figure 6.17, waiting for input if no transition is applicable. As the states are entered, the corresponding *Lisa* bodies are executed. The set of statements, rvalues and conditions in *Lisa* can be extended by implementing new keywords for either in the core framework, or in plugin classes that can be added dynamically.

6.5 Using Planning to Schedule Activities and Games

6.5.1 Motivation and Applicability

In Section 2.5.1.2, we already questioned the adequacy of a fully plan-based approach to dialogue management. During the development of the *VirtualHuman* system, we concluded that the characters would not be able to sensibly plan several moves ahead on the dialogue game level, i. e., as far as the next *utterances* in the conversation were concerned. For one, the set of sensible replies to an utterance in a game is generally small and to a large extent determined by the circumstances (this also fits with the perception of Hulstijn of dialogue games to essentially be small precompiled joint plans (Hulstijn, 2000a)). On the other hand, there are several ways in which responses from other participants can change the interaction by unexpected moves *outside* a current game, e. g., by opening an entirely different topic; but since these responses are a priori not expected, it would not make much sense to plan for all such contingencies beforehand. However, the task level seems much more promising for a planning approach. Also, planning ability would be a valuable addition to the deliberation capabilities of the virtual characters.

In *VirtualHuman*, the planning of the task—or rather, story—progression is managed by the narration engine module. It, however, features a deliberative task that uses planning, namely giving hints to the human player on how to improve the football player lineup in the second phase of the game. The expert character achieves this by computing a sequence of moves that transforms the current lineup to an optimal lineup retrieved from its own knowledge base; here, the plan operators are the possible moves on the football field (placing, removing, and exchanging players). For this, an interface to an external planner was added to the framework (see Section 6.5.3) that was first accessible through setting a goal for a *HardcodedProcess* and subsequently integrated into the *Lisa* language.

In the task-oriented *OMDIP* system, the planning mechanism was used to plan the task level of the interaction. The *Clue* system also uses task level planning, and additionally manages complex coordination of physical actions, e. g., way planning for moving characters, using the planner.

6.5.2 Contingencies

The fact that activities, dialogue games and actions have preconditions and postconditions can be put to use by treating them as operators in a planning algorithm that devises the sequence of games necessary to arrive at a desired world state. However, since the postconditions express only an expectation with respect to what will be the result of executing the game (after all, in the general case cooperation from conversation partners will be required), it is necessary to assess whether the plan's causal links actually hold after each step.

Therefore, a plan for conversation contributions can fail, since there are other interlocutors involved, who might not play along as planned. The definition of an activity states that postconditions are not *guaranteed* to hold after the execution, but only *expected*. During the interaction, unexpected results can occur because the other conversation participants are genuinely non-cooperative, but it is also possible that a commitment to cooperate can make it necessary to disrupt a plan, e. g. because an addressee could not understand an utterance and has to ask again, or interrupts to get more information, etc. Therefore it is crucial to do execution monitoring to determine whether everything goes as planned. In planning, there are two main approaches to this problem (Russell and Norvig, 1995):

- *Contingency planning* tries to account for every possible situation that could arise, and constructs different alternative paths through a plan. During execution, the agent needs to find out what path to select by sensing the actual conditions.
- *Execution monitoring* detects when conditions necessary to continue the course of action do not hold anymore, and revises or reconstructs the plan accordingly. Contrary to contingency planning, dealing with contingencies is deferred until they arise during execution.

Contingency planning including all possible conditions has disadvantages when the set of conditions is large, because then the plan has to include many alternatives and can become exponentially large. If the likelihood for some outcomes is much larger than for others, the agent might be better off to plan for the general case and repair if and when a more infrequent situation arises. For example, if the system asks for the name of a human user, it can be assumed that the user is cooperative and supplies it; if the user refuses to give her name, this can be dealt with as a special case.

6.5.3 Realization

To keep the mechanism as flexible as possible, we aimed for a standard planning interface that allows the framework to utilize different external planners. An accepted standard language for defining planning problems is *PDDL* (**P**lanning **d**omain **d**escription **l**anguage), which is used for problem specification in planning contests, e. g., the International Planning Competition.[9] *PDDL* can be used to specify domains, plan operators and an associated planning problem and is available in several versions from the original *PDDL* 1.0 (Ghallab et al., 1998) to the most recent version 3.0 (Gerevini and Long, 2005). Newer versions add features like temporal and resource constraints to the original language, however, most planners support mainly the *STRIPS* language subset of *PDDL*.

Via the interface, the *JSHOP2* planner was integrated, which is a Java-based adaptation of the *SHOP2* planner. It uses a hierarchical task network (HTN) approach (Nau et al., 1998, 2003).[10] The planner is sent a world description, method and operator descriptions and a goal description in the form of ontological objects and returns a plan for action that can be in terms of activities, games, or single actions. The bulk of the integration work, and the additional implementation of an execution-monitoring process in the framework, was carried out by a research assistant (Gholamsaghaee, 2006) and is not in the scope of this thesis.

6.6 Summary

In this chapter, we described the implementation of the CDE framework to realize our conversational model. The framework is designed to offer high flexibility with regard to the system it is used to implement. The overall size of the core framework implementation, including the GUI components, is approximately 40000 lines of code; the auxiliary classes (e. g., channels) needed for the applications described in the next chapter together amount to another about 15000 lines of code. The computation times of the module are heavily dependent on the particular application and its complexity; however, it generally ranges below one second in all described application instances.[11]

The chapter shows how the framework can be configured by the dialogue designer to create a scenario. We then describe the internal structure of a CDE and the graphical interface that allows to monitor characters in real time. The following section is about how a narration

[9]The International Planning Competition is held bi-annually at the Artificial Intelligence Planning and Scheduling conference series.

[10]HTN planning was used for character behavior planning in other systems, see, e. g., (Cavazza et al., 2002).

[11]Note that computation time may be different from the actual time until a response. In *VirtualHuman*, reactions are artificially delayed since near-instantaneous responses from virtual characters would be perceived as unrealistic by the user.

engine can direct a scenario by setting goals an receiving feedback. A section explained the role of expectations in dialogue games. We describe how processes for the activities are implemented.

We give a motivation for the *Lisa* language, developed for the declarative specification of activities. The structure of a *Lisa* plan, its interpretation and the elements of *Lisa* follows. The next section is concerned with the possibility and use for planning and the integration of an external planner that can be used to aid deliberation for the characters as well as devising the course of action on the task level. In the following chapter, we examine cases where the framework was employed to realize several working systems of differing size and complexity.

Chapter 7

Applications Implemented Using the Behavior Generation Framework

7.1 Introduction

A framework begins to show its real qualities and shortcomings when it is used to implement an actual system. A dialogue system designer may realize a framework that copes with some phenomena in a detailed and theoretically sound fashion, but end up with a system that is only able to handle "toy worlds" because the knowledge engineering task is too complex to handle for larger domains, the inference mechanisms do not scale well enough, or the whole system cannot handle the real-time requirements necessary to provide the users with a smooth interaction experience. Also, for researchers there is the danger of suffering from "institutional blindness": persons who know a system inside out tend to overlook shortcomings that would quickly be discovered by untrained users. Therefore, it is beneficial to get a system out "in the open" as early as possible to be able to get external feedback.

The viability of the CDE framework was shown by using it to implement the dialogue management component of several different multimodal systems, each of which has a different setup, requirements, and scale. In this chapter, we describe how the dialogue management components of the multi-party *VirtualHuman* system, the task-oriented *OMDIP* system, and the multi-character performance *Clue* were realized with the framework. *VirtualHuman* is described most comprehensively, since it is the most complex system and uses most of the features of the framework. For the other systems, we give shorter descriptions and highlight the features that are unique to them.

As already mentioned, the interaction language is German for all systems that we describe here. Dialogue fragments cited in the text are translated to English, while text that appears on screenshots of system outputs remains in German; however, this should not pose any understanding problems.

7.2 *VirtualHuman*

In the course of this thesis, many features of the *VirtualHuman* system have already been presented. We previously gave an outline of the scenario of the football quiz and the lineup game in Section 1.4 of the introduction, and instances of interactions occurred throughout Chapter 5.. Therefore, we restrict the treatment in this chapter to the special modules and their relation to the dialogue manager, and the overall configuration of the system.

7.2.1 Configuration

VirtualHuman runs on three networked Pentium IV computers. A communication software *InfoRouter* acts as a multi-blackboard message system similar to *SmartKom*'s *MultiPlatform* (Herzog et al., 2003): modules can publish output messages of certain types, and subscribe to receive input message types.

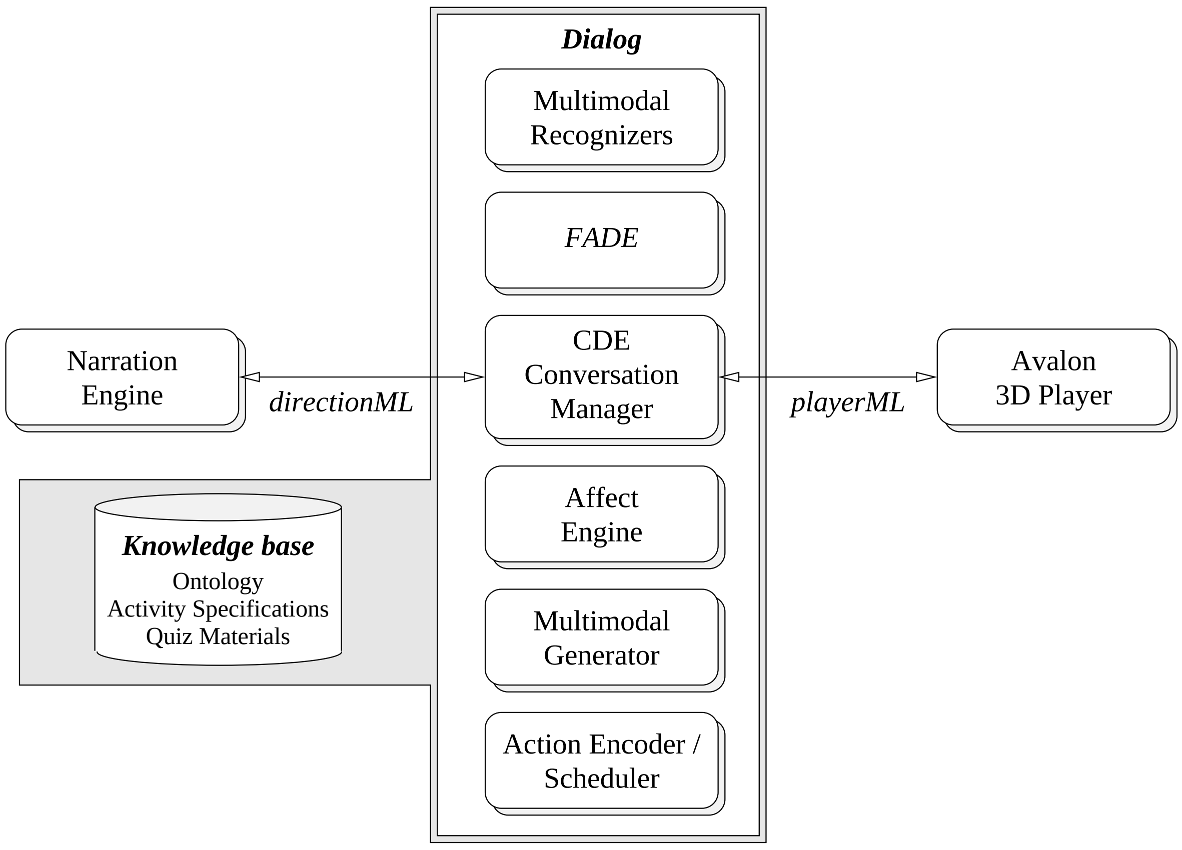


Figure 7.1: The modules in *VirtualHuman*

The system features two separate speech recognizers (one for each human user) with configurable language models, a narration engine module, a *FADE* discourse modeler module, a conversation manager realized with our framework, an *ALMA* affect engine to model the emotional state of the characters, a scheduler called action encoder, and the *Avalon* 3D player that renders the animated scene on a selection of available visualization platforms, e. g., a computer monitor or a 3D projection (cf. Figure 7.1). For user input, there are two tables with mounted microphones and a trackball control (see Figure 1.1 on page 6).

The software configuration related to the dialogue manager comprises several XML files and the ontology. In addition to the configuration resources described in Section 6.2.3, *VirtualHuman* requires a *scene definition*. This is a hierarchical XML structure that contains sub-sections defining the different characters (including gestures, TTS voices, etc.) and the description of the physical scene also needed by the player module. Upon system startup, the dialogue manager informs the player module about the contents of the scene definition. Two special markup languages were developed in *VirtualHuman*. Communication with the narration engine is done via *directionML* (cf. Section 6.3.1.1), and player output and player feedback messages are encoded in the scene language *playerML* (also called *PML*) that is also used for the scene definition (Knöpfle and Jung, 2006). Some parts of the system can also be dynamically configured, i. e., during runtime, by using a GUI interface to the narration engine to let the director character issue commands to other characters, and the system (see Section 7.2.2).

Each character has access to the same ontology. Initially, it was planned to provide each character with its own private version of an ontology, allowing knowledge differing between, e. g., an expert or the moderator. However, the effort of maintaining and developing multiple large ontologies during implementation proved too immense to be handled.[1] To encode different knowledge bases in a single ontology file, we introduced the “Traits” concept to mark instances to belong to some characters only, e. g., detailed knowledge about the players in the German football team is only available to the expert characters.[2]. There are three types of CDEs in the *VirtualHuman* system: Character CDEs, and two kinds of proxy CDEs for users and the narration engine.

7.2.2 Narrative Mode and the Director CDE

The external narration engine module uses an automaton-based approach to set the story goals for the virtual characters while the framework is running in Narrative Mode. Via the goal feedback mechanism, it can adapt the story to what actually happens during the interaction. In the *VirtualHuman* system, we experimented with two different narration engine versions, one provided by *ZGDV* (Göbel et al., 2007) and one developed in-house as an extension to the framework. Both engines use the same control protocol; we use the in-house engine as an example here.

The combination of internal motivation and external narrative control produces semi-autonomous behavior from the characters. The main storyline is determined by the narration module making use of information about the dialogue progress. For example, in Phase 1, the selection of quiz videos is influenced by the performance of the human user. If neither user manages to answer the initial question, the narration engine will decide to tone down the difficulty and subsequently select easier videos to provide a more rewarding experience for the user.

[1]This issue is in our opinion an important area for future research (see Chapter 8).

[2]An instance with a *has_trait* slot can only be used by characters that are defined to have this trait (e. g., the football expert has an “expert” trait)

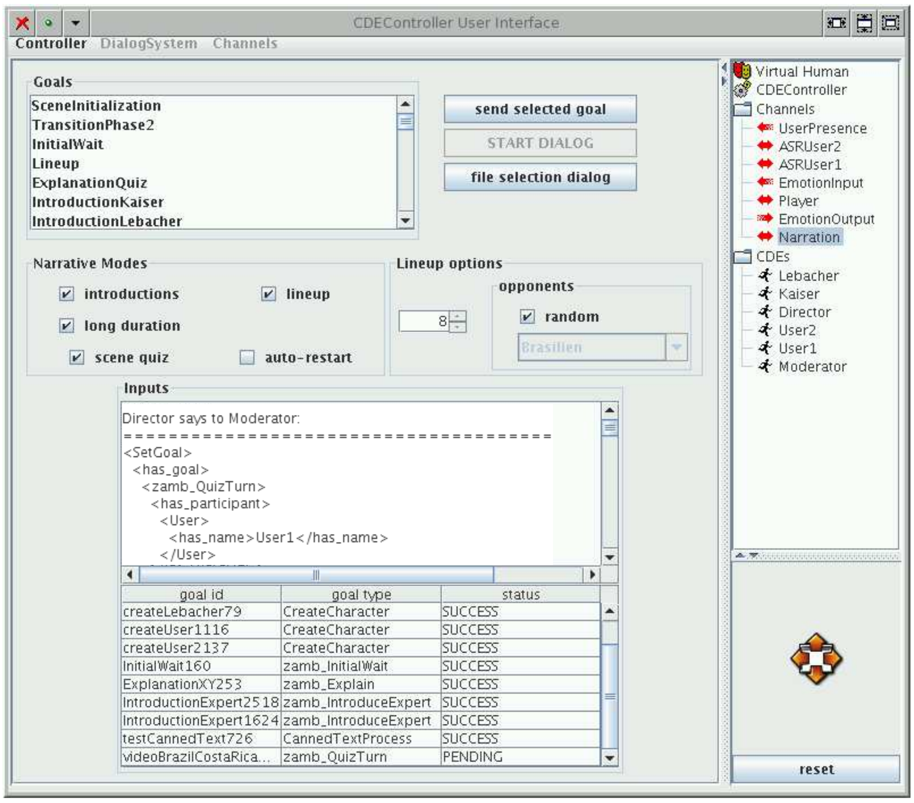
The succession of goals is also chosen in such a way that it is ensured that necessary conditions for the continuation of the story are met. In the quiz stage, an extra question round is inserted if there is a tie in points after the scheduled three rounds, giving no clear winner. If the tie persists even after the additional round, the moderator picks a winner randomly, to prevent the quiz from going on indefinitely. The moderator also comments these decisions to explain and justify them, as in the following example:

MODERATOR: *We have a tie, therefore we will do another round.*

. . . (an additional round is played, but the players still have equal scores afterwards)

MODERATOR: *Regrettably, there still is a tie* [shrugs]. *Since we do not have enough time, I declare user one as the winner of phase one. We will now continue with phase two.*

The fact that the progress of the story is controlled by an external "director" module makes the character agents themselves largely independent of (macro-)narrative concerns. They only need the ability to come up with actions that accomplish their current "stage" directions. Changing the story can then be achieved by adjusting the settings for the narration engine only. To enable the narration engine to exert control over the state and progress of the story, the goal feedback mechanism (see Section 6.3.1.2) is used. If a goal does not yield the desired results that are needed to continue the story, the narration engine can try a different strategy.



Goals
SceneInitialization
TransitionPhase2
InitialWait
Lineup
ExplanationQuiz
IntroductionKaiser
IntroductionLebacher

Narrative Modes
[✓] introductions [✓] lineup
[✓] long duration
[✓] scene quiz [ ] auto-restart

Lineup options
8
opponents
[✓] random
Brasilien

Inputs
```
Director says to Moderator:
=========================================
<SetGoal>
  <has_goal>
    <zamb_QuizTurn>
      <has_participant>
        <User>
          <has_name>User1</has_name>
        </User>
```

goal id	goal type	status
createLebacher79	CreateCharacter	SUCCESS
createUser1116	CreateCharacter	SUCCESS
createUser2137	CreateCharacter	SUCCESS
InitialWait160	zamb_InitialWait	SUCCESS
ExplanationXY253	zamb_Explain	SUCCESS
IntroductionExpert2518	zamb_IntroduceExpert	SUCCESS
IntroductionExpert1624	zamb_IntroduceExpert	SUCCESS
testCannedText726	CannedTextProcess	SUCCESS
videoBrazilCostaRica...	zamb_QuizTurn	PENDING

Figure 7.2: Screenshot of the narration engine interface of *VirtualHuman*

Figure 7.2 shows the part of the controller interface for the director CDE. It is adapted to the possibilities of the narration engine. An operator can choose a set of parameters for a run of *VirtualHuman* at run-time. In the top part of the GUI, an operator can select and send narrative goals manually or switch between the external or built-in narration engine. Below, some controls ("Narrative Modes") allow to set story parameters that are helpful to tune the story to different presentation situations. The verbosity of the story can be adjusted, e. g., the moderator can leave out the introductions of the experts and only give a brief version of the explanations when the system is demonstrated repeatedly, and the human users already know the setting. The timeouts for the lineup phase can also be configured, or parts of the story can be left out if desired (e. g., the quiz phase can be skipped to directly start with the lineup game), and the operator can select a specific opponent for the lineup phase, which allows to adapt the performance to an imminent match in the real world. On the bottom, the GUI protocols the goals sent to and the feedback received from the CDEs, and shows the state of currently executed or finished goals.

During the development of *VirtualHuman*, the possibility to use the proxy CDE, via the interface, to manually simulate the narration engine one step at a time, turned out to be a very valuable tool for testing and debugging the system.

7.2.3 Character and User CDEs

The components associated with *CharacterCDE*s in *VirtualHuman* are shown in Figure 7.3 on the left. The action encoder and the multimodal generator, which are necessary to annotate outgoing dialogue acts with scheduling information and producing surface utterances from semantical representations respectively, were tightly integrated by extending the CDE java class. Discourse history and multimodal fusion are handled by the *FADE* module, which was also embedded. Additionally, the character CDEs are interacting closely with the affect engine *ALMA* and the narration engine.

The *UserCDE* extensions of the CDE class incorporate the gesture and speech recognition and interpretation modules, *FADE*, and no dialogue engine. A user CDE can be disabled by the narration engine without removing it from the scene; this is used when the second expert character is excluded from the second phase.

Like with the director CDE, the GUI interfaces to the character and user CDEs proved very valuable for testing and debugging purposes. The internal knowledge state of the characters can be examined on-line while the system is running, and it is possible to enter speech recognition results directly into the GUI, thereby testing the reaction of the dialogue subsystem, including *FADE*, without requiring any other modules to be on-line (the latter functionality was implemented by Norbert Pfleger).

7.2.4 Realization Scheduling

The passing of communicative action in the abstract representation takes very little time. On the other hand, the user-perceivable actions realized in the environment take much more time to complete. One reason for this is that there are delays to produce the final output (e. g., text-to-speech processing). Additionally, the actions themselves have a duration. Depending

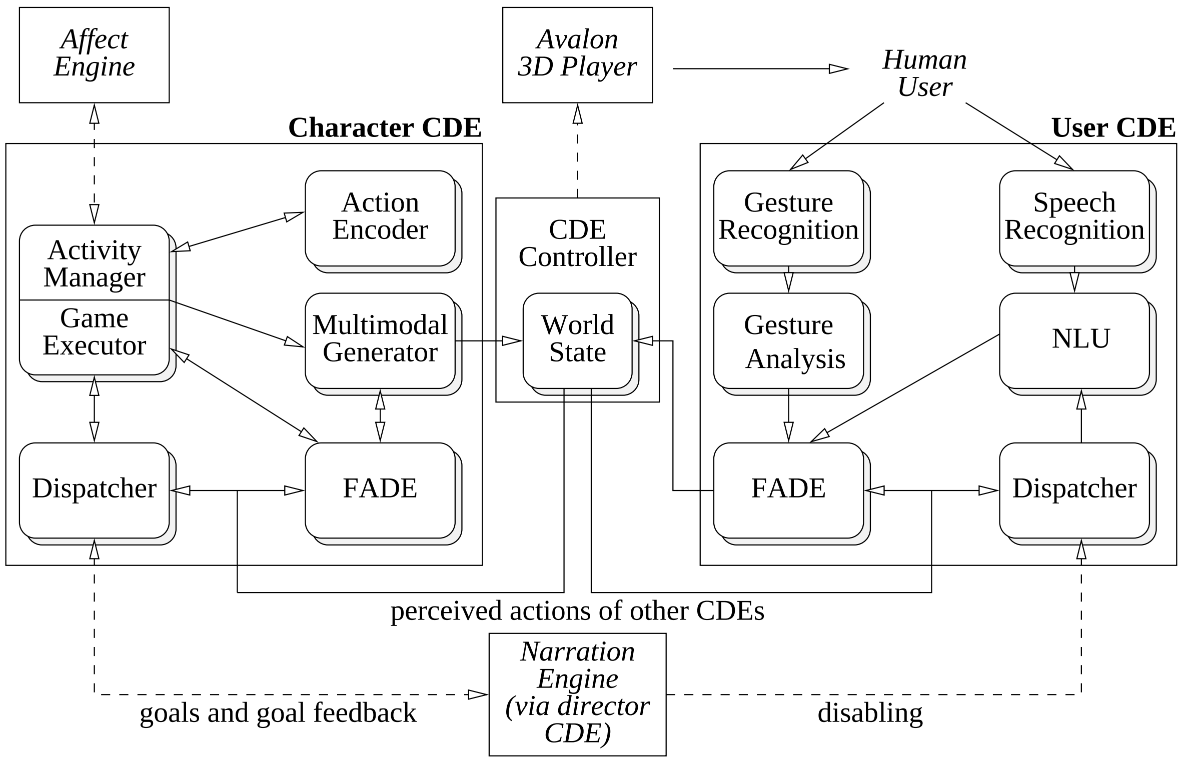


Figure 7.3: Associated components of character CDEs (left) and user CDEs (right) in *VirtualHuman*. Dashed lines indicate translations to external message formats

on the kind of action and other factors (such as cognitive load), the user needs some time to perceive at least some part of the action to understand its meaning. In the case of a gesture, e. g. a pointing gesture that takes one or two seconds, understanding is usually established quickly. In the case of spoken utterances, it is possible that the meaning is understood before the utterance is completely realized; however, if the meaning is complex, it can also be that the user needs extra time to think even after the realization is finished.

In the *VirtualHuman* system, the virtual characters can process dialogue acts in very little time. However, if the acts were delivered to them only after completion, they would not be able to react early to actions that conceivably do not need full realization to be be appraised. We added *realization scheduling* to give the interaction a more natural feel. It uses a heuristic that sends messages containing gesture information immediately after they begin to be rendered. Spoken utterances are sent to the overhearers after two thirds of the time they take to be completed.[3] The realization times are extracted from an action schedule that is computed by the action encoder (Klesen and Gebhard, 2007).

The effect of this is illustrated in Figure 7.4. It shows overlapping acts in a conversation between the moderator and an expert character. The moderator announces a move to *"put (the football player) Ballack in the midfield"*. This statement is delivered to and understood by the expert before it is completely realized in the environment. The expert processes this

[3]This fraction is a heuristic value obtained by experimentation to determine plausible behavior.

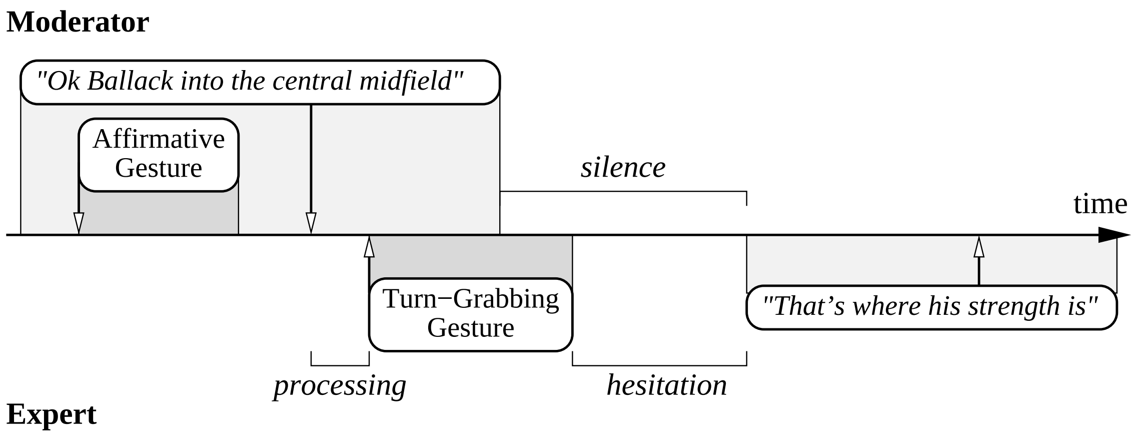


Figure 7.4: Realization time points of overlapping acts. The filled boxes indicate the duration of the realization of an act, the arrows the point in time when the controller passes the act on to the receivers.

dialogue act and, still before its complete realization, decides to comment on the move.

However, the expert is not immediately allowed to speak, since she does not hold the floor while the moderator is still talking. Before an act is sent, the framework calls the floor-management subroutine in *FADE* which generates a turn-grabbing gesture if the turn has not been released (e. g. raising an arm, cf. Figure 7.5). This occurs in parallel to the ongoing utterance by the moderator, and tries again after some time. When the expert eventually gets the turn, she can proceed to utter her comment. If the turn-grabbing does not succeed after several tries, the character may decide to abandon her intention to comment. The realization scheduling mechanism was more thoroughly described in (Löckelt and Pfleger, 2006).

7.2.5 Affective Modeling in Cooperation with *ALMA*

The ontology of *VirtualHuman* defines the concept *CharacterModel*, a subconcept of *ParticipantModel*, whose instances can hold individual traits for virtual and human dialogue participants, and attributes describing their current affective state. In addition to being loaded as static members of the ontology, they can be dynamically updated during the interaction.

The selection of roles in this concept, shown in figure 7.6, matches the character updates sent by the affect engine *ALMA* ("A Layered Model of Affect", (Gebhard, 2007)) that continuously computes a dynamic emotional state for all characters based on weighted affective events combined with general preconfigured personality traits for the characters and periodically informs the dialogue manager about the current mood and the dominant emotion of a character. The emotional state depends on the social relations between the characters, as well on the events that happen in the environment. The personalities of the characters are described in terms of a psychological model using the "big five" characteristics describing a personality in terms of a combination of openness, conscientiousness, extraversion, agreeableness, and neuroticism (Gebhard, 2005; John and Srivastava, 1999).

In addition to the affect attributes, a *CharacterModel* also contains an account for the (static)

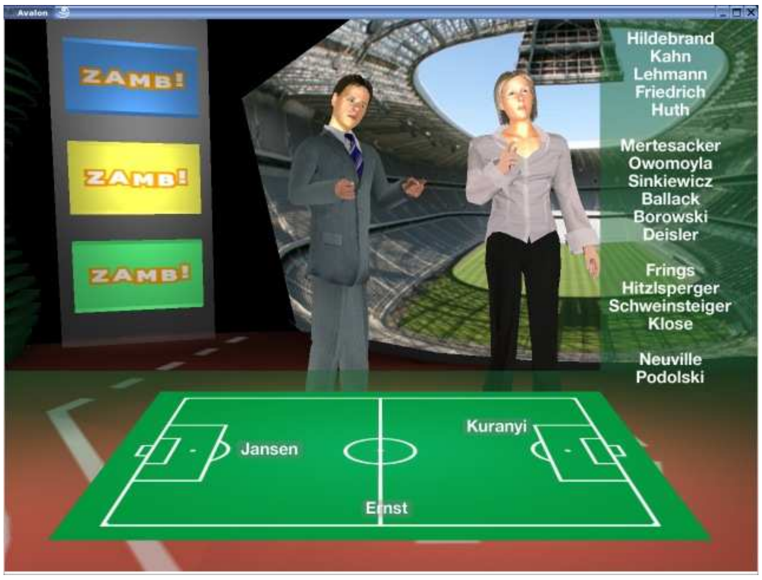


Figure 7.5: The expert (on the right) trying to grab the turn while the moderator is speaking

personality parameters and possibly multiple *Trait* instances that endow the character with additional “attitudes” such as, e. g., talkativeness; this is an easy way to make a character’s behavior configurable by simply changing a the character model instance in the ontology.

To infer the current mood and the dominant emotion of the character, the affect engine needs to be notified of events and their affective impact. During the dialogue, the dialog manager generates affective events that match the content of the utterances and update *ALMA* accordingly. There are events that are generated in the executable body of an activity state (e. g., an expert reacts positively if the user asks her for advice) or from annotations in dialogue acts from the ontology. An example for the latter is a remark from an expert like

EXPERT HERZOG: *Mister Kaiser has no idea what he is talking about [GoodActSelf 1.0].*

When this utterance is generated, the *GoodActSelf* tag is extracted and sent as a message to *ALMA*, along with the numerical weight indicating the strength of the affective action. On the other hand, when Expert Kaiser perceives the utterance, he himself will send a corresponding event *BadActOther* to the affect engine.

If the affective state changes, the dialogue manager in turn receives asynchronous messages from the affect engine *ALMA* that update the character models of the respective CDEs. *ALMA* also parameterizes the appearance of the character player, e. g., by changing its idle gestures and facial expression, as shown in Figure 7.7. This in turn can influence the behavior of the characters. In the example, the *ALMA* will send a new state for Expert Herzog reflecting her boosted self-esteem, and one to Expert Kaiser that expresses that he is now angry. In the simplest case, the ontology provides alternative versions of utterances annotated with *Trait* instances corresponding to different moods and personality traits of the character. The character model can also be employed directly in conditions to influence decisions, e. g., a character may refuse an answer because it does not like the character that posed the question, or simply because it is in a bad mood.

CharacterModel	
HAS_INTELLIGENCE	*Integer*
HAS_OPENNESS	*Integer*
HAS_CONSCIENTIOUSNESS	*Integer*
HAS_EXTRAVERSION	*Integer*
HAS_AGREEABLENESS	*Integer*
HAS_NEUROTICISM	*Integer*
HAS_DOMINANCE	*Integer*
HAS_AROUSAL	*Integer*
HAS_PLEASURE	*Integer*
HAS_MOODWORD	*String*
HAS_MOODINTENSITY	*Integer*
HAS_DOMINANTEMOTION	*String*
HAS_DOMINANTEMOTIONVALUE	*Integer*
HAS_TRAIT*	*Trait*

Figure 7.6: The *CharacterModel* concept

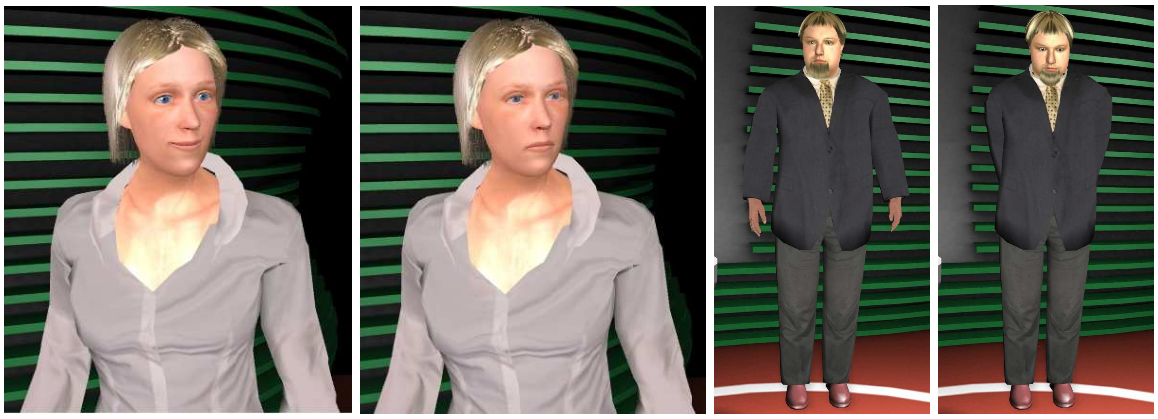


Figure 7.7: Different affective states: (left) Miss Herzog changing from "happy" to "disappointed" facial expression; (right) Mister Kaiser changing from "relaxed" to "docile" body posture.

7.2.6 The Game Logic for the Lineup Game

In phase two of the quiz, an additional process implements a game logic that produces evaluations of the current team lineup and can generate proposals for moves. It is implemented as a singleton object and is shared between the moderator and the expert character. However, it produces different evaluations based on the traits of the character requesting them (the expert is more conservative than the moderator and prefers more defensive teams). The evaluation is based on the ontological descriptions of the players, and the opponent the team is to play against. The game logic can be viewed as an external application that can serve requests by the characters.

- **Move Scores**

  Opponent teams are represented in the ontology together with a rating of *aggressive*, *defensive*, *conservative* or *robust*. The *FootballPlayer* concept includes scores for different football-related aspects, like fitness and preferred side to play on.

Striker	
has_firstName	*Miroslav*
has_lastName	*Klose*
has_duelSkill	*77*
has_fitness	*92*
has_headerSkill	*93*
has_shotSkill	*83*
has_technique	*81*
has_side	*center*
has_preferredFoot	*right*
. . .	

Figure 7.8: A *FootballPlayer* instance from the *VirtualHuman* ontology

FootballPlayer also has the subconcepts of *Defender*, *Midfielder*, *Striker* and *Goalkeeper* (part of a sample instance is shown in Figure 7.8). In combination, the exact ontological class of the player, the ability scores, and the preferred strategy against the opponent determine how a player is rated at a particular position. Move scores are different for the moderator and the expert: the moderator is more cautious than the expert. For example, the moderator has a "cautious" personality and prefers a robust strategy (more defenders) against an aggressive opponent team, while the expert will also counter with an aggressive strategy (more strikers).

- **Proposing Moves**

  The game logic assembles an "ideal" team given the knowledge of the female expert and the opponent team. If the expert is asked to propose a move during the lineup game, the game logic assumes the current team consisting of players set by the user as an initial world state and the ideal team as a goal world state and computes a plan consisting of the moves that would be necessary to transform the current team to the ideal team. The first move of this plan is proposed to the user.

  The proposal by the expert is overheard by the moderator character, and if the user agrees, FADE can resolve the reference to the proposed move, and the moderator can immediately execute the move (cf. turns (3)-(8) in the the example of Section 5.7.2, page 154). The moderator then updates the score. Because of the different personalities of moderator and expert, it can happen that a move proposed by the expert does not get a good score from the moderator.

7.2.7 Dynamic Help and Explanations

During the narrative, an auxiliary process offering the service of *Dynamic Help* is continuously running as a background process in the moderator's CDE. It engages in dialogue games that start with requests of the user for instructions, such as *"What can I do here?"*. If the user makes such a request, the process determines the game phase by examining the moderator's other running processes. It then looks in the ontology for instances of type *ActivityProcess*

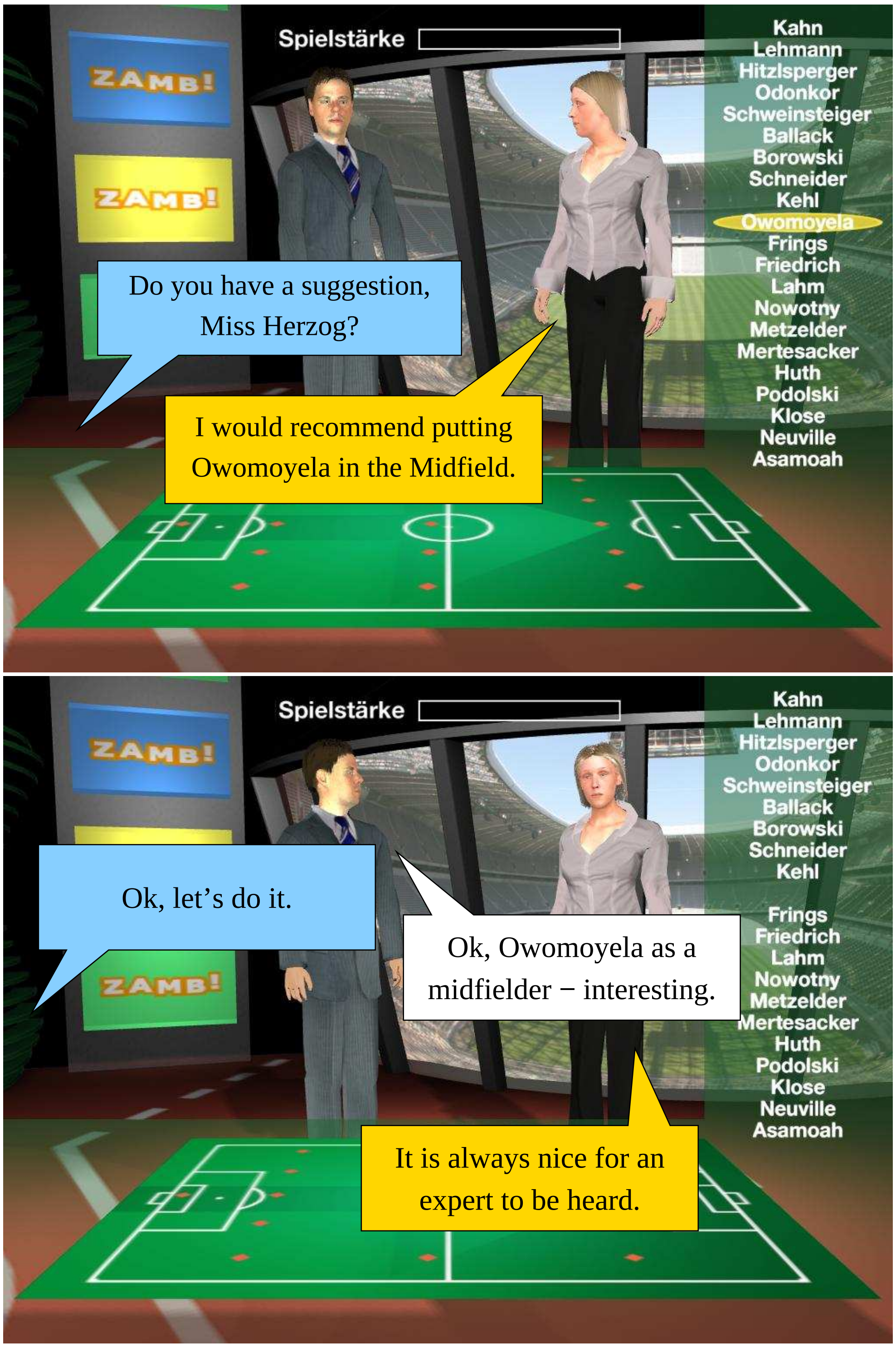


Figure 7.9: Requesting and accepting a move proposal by the expert

that have available affordances of type *Explanation* containing a canned help text. If such an affordance is found, the user is given this explanation. Explanation affordances are present for most activities in the *VirtualHuman* scenario. A possible dynamic help instance would be

(1)	USER:	*What can I do?*
(2)	MODERATOR:	*You might ask Mrs Herzog for her opinion. She knows most players personally.*

Explanations are also generated from other attributes of the objects. If the user asks the female expert about her opinion of a particular football player, she uses the most specific type the player belongs to (which could be one of *Goalkeeper*, *Striker*, and so on) and information stored in the instance representing the player, which can include, e. g., the preferred foot, the side the player prefers, or various fitness values. For additional atmospheric value, the instance in some cases contains additional comments of stereotypical football expert talk. From this, the expert constructs a comment like

(1)	USER:	*What do you think of Huth?*
(2)	EXPERT:	*Robert Huth normally plays as a defender and prefers the right foot. He is not very fast, but nevertheless difficult to overcome. I would place him in a central position.*

Another possible use of annotated objects used in the first prototype enabled the construction of an explanation of the game in variable level of detail. For this, the annotations in the objects to be explained were given causal links that connected the object to be explained with other objects that could be used to elaborate on the explanation.

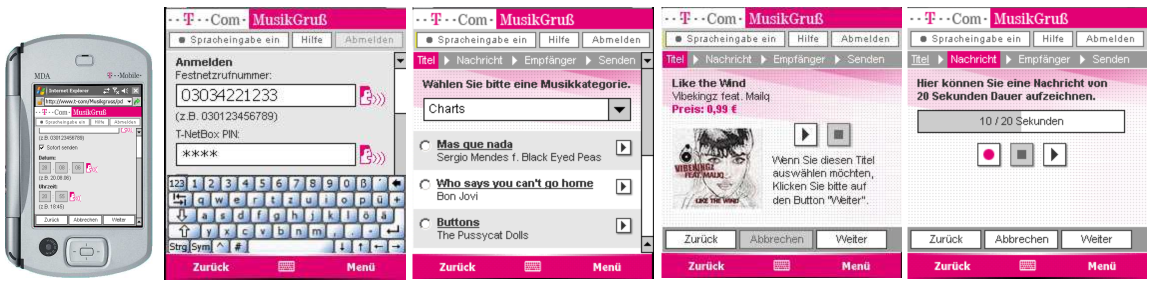
7.2.8 Timeout Reactions

To keep the flow of the quiz and lineup games going, the moderator character also has the role to incite hesitant users to contribute to the interaction. If a user remains silent for too long, the moderator CDE will switch to an uneasy affective state and either produce encouraging remarks (as exemplified in, e. g., move (11) of Figure 1.5 on page 9) , or autonomously suggest actions to the user. Since the second game phase has a time limit for the duration of the lineup assembly, the moderator keeps track of its progress and will announce when, e. g., half the time is up, or the game is soon to end. The time limit is set dynamically by the narration engine, and the moderator will also signal longer phases of user inaction to the narration engine. *VirtualHuman* can be configured to suppress the moderator's autonomous suggestions and leave it up to the narration engine to trigger appropriate actions explicitly.

7.3 *OMDIP* and *Clue*

7.3.1 *OMDIP*

OMDIP is a prototype for a small, task-oriented multimodal system that allows a single user to interact with a web-based application from a handheld device. There were two phases of the project is with different underlying application scenarios.



Wählen Sie bitte eine Musikkategorie.

Wenn Sie diesen Titel auswählen möchten, Klicken Sie bitte auf den Button "Weiter".

Hier können Sie eine Nachricht von 20 Sekunden Dauer aufzeichnen.

Figure 7.10: The *OMDIP* device and some sample screens from the application (in German)

The first scenario allows the user to purchase soundlogos for a phone,[4] the second, somewhat more complex scenario is about composing a musical greeting message, i. e., a spoken recorded audio accompanied by a musical background, e. g., an up-to-date chart song. The user can use combined speech and stylus input to order and configure soundlogos for mobile phones and compose musical greeting messages via a multimodal user interface. The handheld device communicates with a server application logic that provides the dynamic data; the graphical presentations are rendered as Java Server Pages (JSP) in Internet Explorer (see Figure 7.10). Overall, the application comprises nine different screens; in principle, the user can move more or less freely between the screens, although there are restrictions conditional on the task state.

The system uses the *IHUB* communication infrastructure that has been adapted from the *SmartWeb* project (cf. Section 3.2.8). Its component modules and their relations are shown in Figure 7.11. *OMDIP*, too, is based on a full ontological domain modeling. As the base ontology, the *SmartWeb* ontology (Sonntag and Romanelli, 2006) was used, which already includes an comprehensive coverage of concepts for the representation of different media types, including musical items. It was extended with additional concepts of additional domain knowledge and the task knowledge needed for the required activity types. The dialogue management branch of the *VirtualHuman* ontology was merged with this base ontology, making the interaction with the CDE framework possible.

Even though *OMDIP* does not realize true multi-party dialogue, the human user and the system itself are represented by one CDE each, and the system could easily be extended to support multiple users, or more than one system agent. The activity specifications for *OMDIP* were written exclusively in the *Lisa* language. The system comprises 13 activity types, which register 88 services in total. The activities in *OMDIP* are annotated with preconditions and postconditions to make them usable as plan operators. The system uses the JSHOP2 planner interface created during the *Clue* project (see next section) to dynamically plan the activity sequence.

The selection of available background songs is delivered dynamically from the external database. This way, the system can, e. g., feature up-to-date music charts. Since the musical pieces can be identified by the user via speech, the ASR language model is updated with dynamical lexicon updates via the expectation mechanism each time a song list is retrieved. *OMDIP* acts as a front-end between the user and an external web application. The dialogue

[4]A soundlogo is an audio snippet that is played for the caller as a replacement for a signal tone.

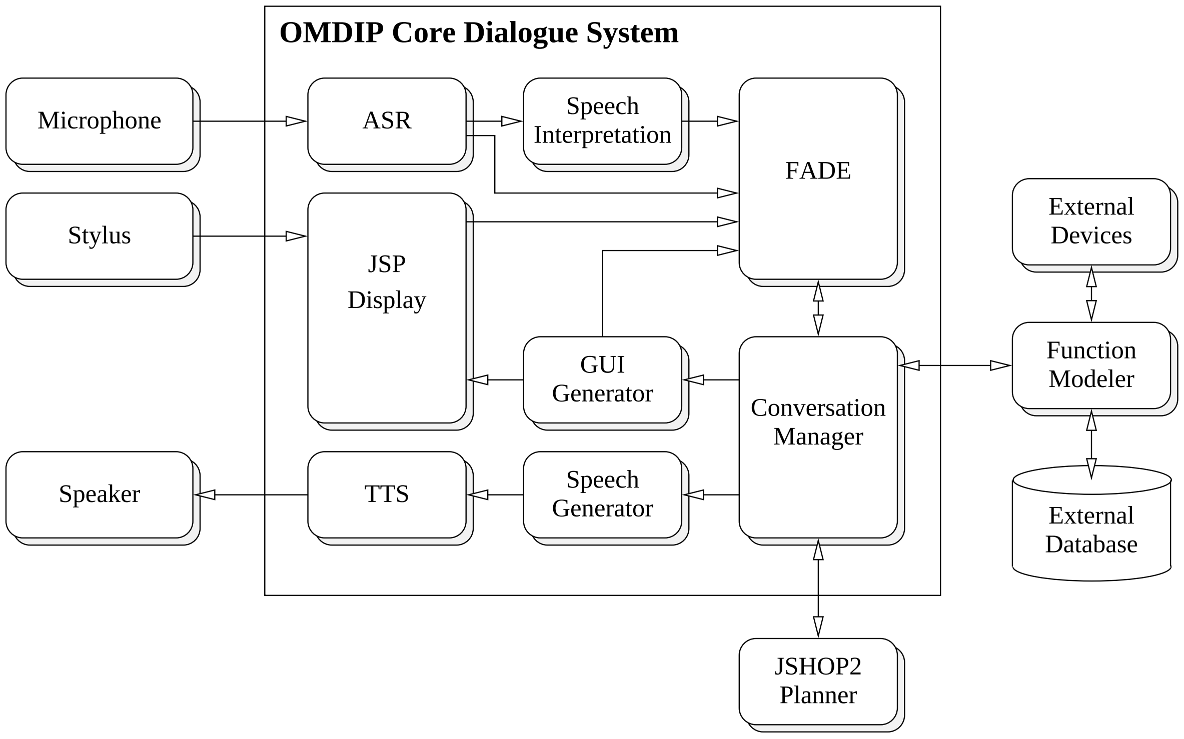


Figure 7.11: The *OMDIP* architecture

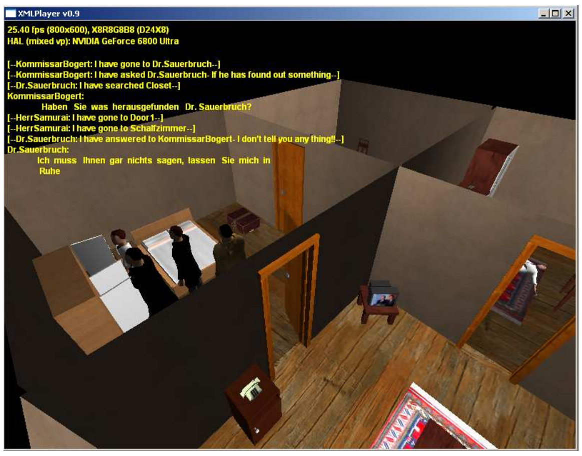
agent communicates with a function modeller using dialogue games implementing the application protocols. Most of them are *Request-Response* games where the system takes the initiative, but there also are some cases where the application takes the initiative to, e. g., asynchronously *Inform* the system about the completion of an operation.

7.3.2 *Clue*

The *Clue* system uses the CDE framework to stage a performance with multiple virtual characters in a mystery game setting modeled after the well-known board game *Cluedo*.[5] It was realized as an advanced programming project by three participating students. The CDE framework is coupled with a narration engine and a 3D player that was specificially developed for the project. *Clue* does not feature user interaction (apart from the possibility to move, zoom and rotate the player view).

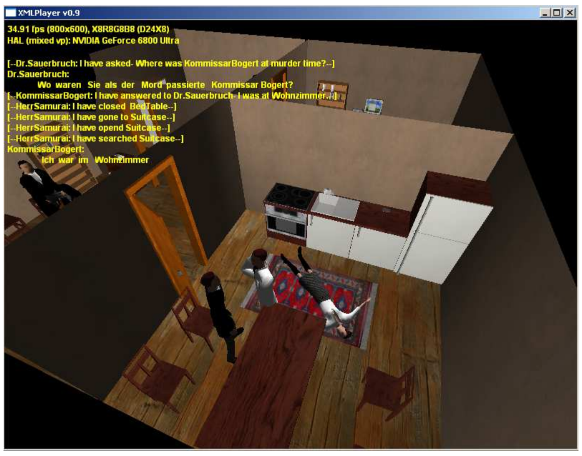
Each student was assigned one of three largely independent subtasks for the project: (1) designing and encoding the story flow and implementing a narration engine, (2) interfacing the *JSHOP2* planner to the framework, and specifying the planning knowledge sources, and (3) developing an XML-controllable real-time 3D player including the modeling of the characters and objects, and combining it with the existing *Mary* TTS system (Schröder and Trouvain, 2003) for speech output. The base ontology was *VirtualHuman*'s; the students with the tasks

[5]Incidentally, the board game has also been renamed *"Clue"* in the United States, and there also is a *Cluedo*-based computer game of the same name.



```
25.40 fps (800x600), X8R8G8B8 (D24X8)
HAL (mixed vp): NVIDIA GeForce 6800 Ultra

[--KommissarBogert: I have gone to Dr.Sauerbruch--]
[--KommissarBogert: I have asked Dr.Sauerbruch- If he has found out something--]
[--Dr.Sauerbruch: I have searched Closet--]
KommissarBogert:
        Haben  Sie  was  herausgefunden  Dr. Sauerbruch?
[--HerrSamurai: I have gone to Door1--]
[--HerrSamurai: I have gone to Schalfzimmer--]
[--Dr.Sauerbruch: I have answered to KommissarBogert- I don't tell you any thing!!--]
Dr.Sauerbruch:
       Ich  muss  Ihnen  gar  nichts  sagen,  lassen  Sie  mich  in
        Ruhe
```



```
34.91 fps (800x600), X8R8G8B8 (D24X8)
HAL (mixed vp): NVIDIA GeForce 6800 Ultra

[--Dr.Sauerbruch: I have asked- Where was KommissarBogert at murder time?--]
Dr.Sauerbruch:
       Wo  waren  Sie  als  der  Mord  passierte  Kommissar Bogert?
[--KommissarBogert: I have answered to Dr.Sauerbruch- I was at Wohnzimmer.--]
[--HerrSamurai: I have closed  BedTable--]
[--HerrSamurai: I have gone to Suitcase--]
[--HerrSamurai: I have opend Suitcase--]
[--HerrSamurai: I have searched Suitcase--]
KommissarBogert:
        Ich  war  im  Wohnzimmer
```

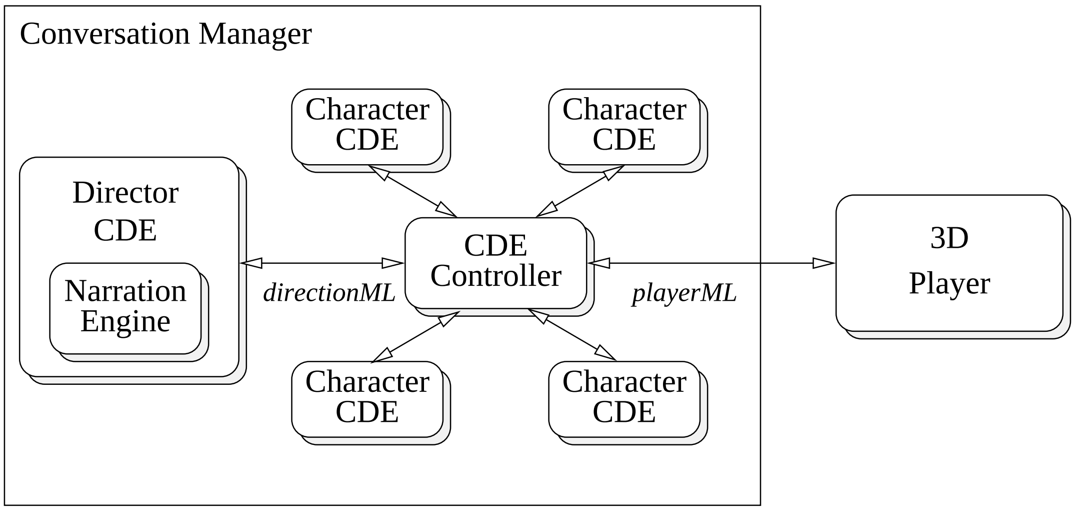


Figure 7.12: Screenshots from *Clue* with some trace messages (German realizations of the utterances included); *Clue*'s module setup

(1) and (2) jointly extended it to include the necessary domain and task knowledge. Figure 7.12 shows two screenshots from a *Clue* performance and the setup of the system. *Clue* runs in Narrative Mode. Its narration engine is, unlike the one in *VirtualHuman*, not realized as a separate module, but was integrated into the director CDE. The narration engine in *Clue* uses an approach based on a finite state automaton whose transitions are conditional on the goal feedback messages from the characters.

The physical environment consists of four interconnected rooms (kitchen, living room, bedchamber, and study) with various pieces of furniture. Possible murder weapons and other evidence are hidden in various places in the scenario, including inside closed objects. The yellow text in the upper left corner is a live protocol of the actions and utterances of the characters (the utterances are made in German language). The goal of the (innocent) characters is to cooperate to find the evidence and to combine it to be able to derive who is responsible for the murder, and to determine the motive and the murder weapon. The murderer, unsurprisingly, has the opposing goal to hinder the solution of the mystery by hiding evidence and telling misleading stories.

The characters can be instructed to employ different strategies for solving the criminal case that influence their preference between different actions. The *searching* strategy means that the character prefers to wander through the surroundings to retrieve pieces of evidence. The *asking* strategy inclines the character to ask others about what they know (what evidence they have found, what they know about the other characters, and what their inferences are). A *commanding* character gives instructions to others to e. g., go look for the murder weapons. The characters also have different degrees of cooperativity, which determines their tendency to answer or obey other characters. The design of the narrative component was further described in (Nikolova, 2006).

The interfacing of the *JSHOP2* planner to the CDE framework was part of the work done by one of the students and re-used in *OMDIP* for planning on the task level, as mentioned in the description of the *OMDIP* system. In *Clue*, planning was not restricted to the task level, rather it was employed to devise the entirety of the character actions including physical actions. In total, *Clue* features 21 operators for physical actions and 31 operators for communicative actions. The details of the planner adaptation are described in (Gholamsaghaee, 2006).

7.4 Summary

This chapter described three dialogue system instances that were realized with the conversational behavior generation framework: *VirtualHuman*, *OMDIP*, and *Clue*. The systems are substantially different in scope, theme, and purpose, and each one has unique features that had to be accounted for.

- *VirtualHuman*, the most complex example, is an interactive, multimodal, multi-party storytelling system that uses the Narrative Mode of the framework to coordinate the actions of three virtual characters with individual traits that interact with two human users. The scenario requires solutions with respect to synchronizing turn-taking and overlapping contributions, fast real-time reactions, cooperation with affective modeling and story-introduced constraints like changing scenes with differing character combinations and timeouts.
- *OMDIP* is a task-oriented system for one user that interacts via dynamically created JSP pages on a web browser. The system interfaces via a function modeler with a database backend that provides the content for the modeled task. Like the presentation mode, the system infrastructure is quite different from *VirtualHuman*: For the knowledge base, the *SmartWeb* ontology was re-used and adapted; the communication infrastructure employs *SmartWeb*'s *IHUB*.
- *Clue* is a student project that demonstrates that the framework can be used, with some training, by non-experts to create a small yet functional and nontrivial storytelling system. It features multiple characters that, beyond communicative interaction, also use a variety of physical actions to perform a story with several possible endings.

The selected systems demonstrate the expressive capabilities of the conversational behavior generation model and the versatility of the implementing framework to adapt to a variation of different task types and requirements.

Chapter 8

Conclusion

We have presented a way to model the knowledge base, a dialogue model for multimodal and multi-party conversations with mixed virtual and human participants, and a framework implementing the model. We then described how it these components are used to realize the conversation manager for three dialogue systems that exhibit substantially different requirements, scale, and setup. The framework has been shown to be flexible enough to employ different paradigms for devising action, including processes written in a general purpose programming language, external planning algorithms, and action plans written in *Lisa*, an extensible language for the special purpose of specification of dialogue activities.

8.1 Contributions

The contributions of this thesis comprise scientific results in the field of multimodal multi-party conversation management, practical contributions in form of an implemented framework and applications realized with it, and a number of publications that arose from the research. The following sections gives an overview of these contributions.

8.1.1 Scientific Results

The main scientific results of this thesis are the following:

- **Modelling sophisticated goal-oriented and cooperative multi-party conversation**

  A three-layered model of dialogue acts, dialogue games and activitites was proposed. It describes joint conversational action by cooperating conversational dialogue engines (CDEs) representing the participants of multi-party dialogue.

  - The model achieves natural coordination of emergent multi-character interactions. An arbitrary number of activities can be pursued in parallel and still interact with each other.
  - A mechanism and a protocol for *Realization Scheduling* was implemented that coordinates the realization of interactions to appear more life-like.

  - *SmartKom*'s expectation representation was adapted to the ontological modeling and improved to describe more expressive templates for communicative actions instead of expected slots. Using expectations, the conversation manager can provide other modules with disambiguation help.

- **Construction of an adequate knowledge base for modeling multi-party conversation**

  A comprehensive modeling of the knowledge resources for conversation management was designed and realized in a standardized ontology format. With respect to the different types of knowledge outlined in Section 4.2.2, the taxonomical ordering, world and domain knowledge, this was joint work with the collegues responsible for other modules in the systems, especially the *FADE* module, with which the conversation manager shares a common representation. Here, the main contribution of this thesis is the modeling of the conversation management branch of the ontology, which features concepts relating to dialogue acts, dialogue game rules, and activities, and the task knowledge for the different activities of the systems. The dialogue branch, originally created in the *VirtualHuman* ontology, was shown to be sufficiently generalizable as it was transferred and re-used in the *OMDIP* knowledge base derived from *SmartWeb*'s ontology.

- **Specification of autonomous and semi-autonomous behavior of virtual characters**

  The behavior model of the virtual characters can exploit the full inferential power of the ontological representation. It allows for independent autonomous conversational and "physical" action for each character, or semi-autonomous action under the supervision of an external module for narrative control. The characters also support dynamic affective modeling by an external module that can influence their behavioral choices. They can pursue multiple independent or related goals in parallel.

  - The character's behavior specification can be specified in a wholly declarative fashion. Dialogue acts and dialogue games reside entirely in the ontology, while activities additionally use plans specified in the *Lisa* language, or dynamically generated by an external planner module.
  - The characters can act with regard to internal goals, external goals or a combination of both. The *directionML* protocol allows a fine-grained control of external goals and feedback of their results.

- **Providing a method for incremental construction of conversation specifications**

  The three-layer model uses a set of building blocks that is easily extendable and reusable across applications. The dialogue games from the existing applications provide a solid base that can be enhanced either by extending existing or constructing new ones, while activity specifications are largely application-specific.

  - The development of a dialogue system can be done incrementally by adding one activity at a time to a functional system.
  - The system remains highly configurable, since behaviors are located in description units that can be easily and independently added or removed.

- **Allowing for multiple paradigms for activity specification**

  The behavior of the characters can be defined by explicit implementation in Java classes, or by specifying it in the *Lisa*, a language that was designed for activity specification. In addition, an external planner can be used on the task level.

  - Although planning could also be used on the dialogue level (as was done in *SmartKom*), we argue that this is neither very useful nor realistic, since the planning units are too small: in a typical conversation, it is simply not possible to plan several moves ahead. This restriction does not impair planning on the task level.
  - The process of designing *Lisa* was aided by analyzing activities written in Java and identifying code patterns that frequently occurred, like manipulation of slots, matching ontology content, and constructing utterances. *Lisa*'s language elements correspond to such patterns, and the language remains extendable.

8.1.2 Practical Contributions

- **Providing a framework that implements the conversational behavior generation model**

  A unified, configurable and reusable behavior generation framework was provided that can manage real-time interactions involving an arbitrary number of virtual characters and human users. A conversation manager coordinates an individual conversational dialogue engine for each participant. The framework allows for the flexible integration of external modules and applications, and additional tools for testing and monitoring. A generic process allows adaptation of PDDL-conformant planning algorithms to the system. The procedural aspects of activities can be developed in a rapid-prototyping fashion.

- **Support for Narrative Mode**

  In addition to autonomous behavior generation, the framework is also capable of running in (semi-autonomous) Narrative Mode. In this configuration, story development is directed by a dedicated external narration engine via fine-grained control of goals for characters and character groups in combination with a feedback mechanism. This control mechanism is made available via the *directionML* protocol.

- **Evaluation by implementation and deployment of applications**

  The framework was evaluated in practice by putting it to use to implement and deploy three multimodal conversation applications of varying domain scale from student-project sized to very large, and of different thematical background.

  - In case of the largest and most complex instance, *VirtualHuman*, it was the first time that comprehensive ontological domain modelling for an application, multiple life-like characters communicating with multiple users in real-time, multidimensional expressive modalities (speech, gestures, facial expression, body posture, ability to interact physically with a virtual environment), as well as support for external affective modeling and dynamic narrative control were integrated in a single system.

  - *OMDIP* showed that the framework can also be used for developing a web-based task-oriented system with ontological domain modeling, and that it is possible to accommodate general-purpose ontologies originally developed for different systems and adaptable to other communication infrastructures.
  - *Clue* is an example for a smaller scale project, and demonstrated that non-experts (three students) could, with some training, re-use and utilize the conversation manager and other resources from *VirtualHuman* to build a full-fledged, real-time mystery narrative.

- **Facilitating ontology access and manipulation**

  Efficiency issues in ontology manipulation with common RDF(S) libraries like *Jena* were addressed by providing a lean interface, *JenaLite*, that lets the user access and modify the ontology in terms of an intuitive, TFS-based API view and exhibits increased performance. This comes at the price of sacrificing some of the features of RDF(S), which however—in contrast to the improved real-time performance—were not required for our task.

8.1.3 Publications

This section gives a list of the publications that resulted from the research for this thesis.

- **Journals and Book Chapters**

  Parts of this work have been published in the book *Smartkom: Foundations of Multimodal Dialogue Systems* (Wahlster, 2006) (see (Löckelt, 2006)) and in the International Journal of Virtual Reality (Löckelt et al., 2007). An extended version of (Pfleger and Löckelt, 2006) is to appear as a chapter in the book *Fusing Intelligence in Virtual Agents* (Jain, 2008).

- **Conferences**

  Parts of this work have been presented at the following international conferences: *Konferenz zur Verarbeitung natürlicher Sprache (KONVENS)* (Löckelt, 2004), the *International Conference on Virtual Storytelling (ICVS)* (Löckelt, 2005), the *International Conference on Intelligent Virtual Agents (IVA)* (Pfleger and Löckelt, 2006), the *International Conference on Intelligent Technologies for Interactive Entertainment (INTETAIN)* (Löckelt et al., 2005), the *International Conference on Multimodal Interfaces (ICMI)* (Reithinger et al., 2003, 2006), the *International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE)* (Löckelt and Pfleger, 2006), the *International Conference on Virtual Storytelling (ICVS)* (Kempe et al., 2005), and the *European Conference on Speech Communication and Technology (Interspeech/Eurospeech)* (Pfleger and Löckelt, 2005).

- **Workshops**

  Parts of this work have been presented at the following international workshops: At the workshop series *Semantics and Pragmatics of Dialogue* (Löckelt et al., 2002; Löckelt and Pfleger, 2005), the *IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue*

Systems (Porzel et al., 2003), and the *Workshop on Scalable Natural Language Understanding (ScaNaLu)* (Alexandersson et al., 2004a).

8.2 Future Work

The model and framework can be utilized and extended to further improve multimodal multiparty interaction. To conclude this thesis, we will briefly point out some interesting avenues for further research:

- **Multiple conversation thread management**

  An area where additional refinements would be especially beneficial is the management of multiple conversations. The model and framework do not provide for flexible and sophisticated task switching. While our approach does work adequately for the applications that we have realized to date, it can run into problems for conversation arrangements where different sub-conversation threads or entire goals need to be freely and independently suspended, continued, and aborted.

- **Character groups**

  In the current form, conversation participants can be addressed as groups either by explicitly enumerating them, or by way of underspecified ontology instances (e. g., *Character* instances that have an *Expert* trait, or instances of the subconcept *User*). However, this mechanism leaves much to be desired for several reasons. The deficits include (but are not limited to): (1) it is too dependent on the way the ontology is modeled, and on the knowledge bases of the individual characters, (2) it does not support groups that are heterogenous, or have internal individual group roles, and (3) while possible, it is at best very awkward to form or break up dynamic groups (by manipulating the knowledge bases of the characters, e. g., asserting and retracting ad-hoc traits).

- **Advanced tool support for knowledge engineering**

  Knowledge engineering was still quite difficult and time-consuming with the tools that were available. One reason for this is that general-purpose ontology editors do not cater to the special needs of the kind of knowledge base we had to maintain. One reason for this is that they tend to view ontologies as immutable data structures, while the knowledge base of a virtual character can change dynamically. Another is lack of graphical support for certain dependencies (e. g., the dialogue game graphs could not be visually edited), automated validity checks of the knowledge base, and poor support for entering large amounts of data. We also mentioned in Chapter 7 that given the available tools, it is extremely cumbersome to keep differing but compatible knowledge bases for several characters synchronized.

  These points should not be mistaken as a criticism to mean that the available ontology editors were incomplete or poorly implemented. Instead, we want to emphasize the need for special-purpose tools, perhaps in the form of plugins for editors such as *Protégé*, that can be adapted to support the particular needs of the designers of larger dialogue system ontologies.

- **Multiple-typed dialogue acts**

  Another issue that has not been addressed in the model is that the mapping of utterances to dialogue acts can be ambiguous, or even multi-valued. We assume in this thesis that the discourse modeler is able to assign a unique dialogue act type to any utterance, which, however, is not always possible. A mechanism for handling such ambiguous or multi-valued acts would be a useful extension of the model.

- **Development and quality assurance tools**

  The setup can be exploited to allow automatical and semi-automatical testing of a developed system without additional external tools. This can be done by replacing user CDEs with special character CDEs, which could be called *testing* CDEs, that are designed to carry out the contributions of the user in dedicated use cases or by trying out possible interactions in some systematical fashion. The testing CDE(s) can also verify the system's reactions and produce appropriate conversation and error logs. Such a setup, used in concert with unit testing on the lowest level, and integration testing on the level of the system components, promises to be very useful to the dialogue designer in developing and debugging a new system.

Appendix A

Lisa Specification

As described in Section 6.4, a plan defines an activity by specifying services offered by the activity, slots that hold relevant data, a body for the activity and games that implement the services.

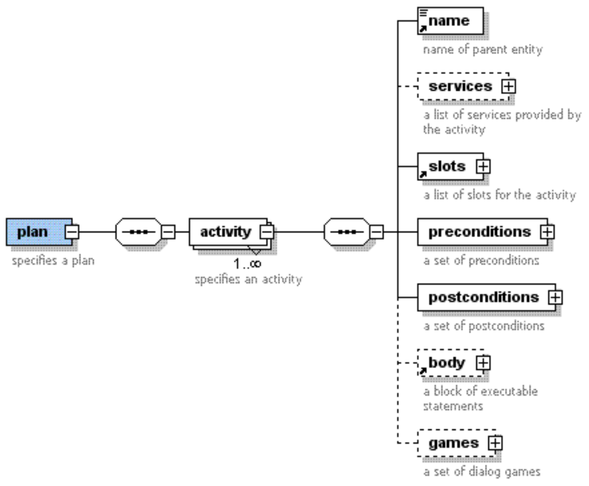


Figure A.1: Toplevel structure of a *Lisa* plan

Figure A.1 shows the toplevel structure of a *Lisa* plan.[1] It contains the *name* of the plan, a set of *services*, a set of definitions for *slots*, preconditions and postconditions for the execution, a *body* of *Lisa* statements that is executed once when the activity is started, and a set of definitions of *games*. The *name* element is a simple string; the other elements are described in more detail in the following sections.

If an error occurs during the execution of a *Lisa* plan, a *RuntimeException* is raised, which terminates the currently running activity with a *failure* state.

[1]The syntax diagrams were created using the XSD schema design view of XML Spy 4.3 from Altova, Inc.

A.1 Service Elements

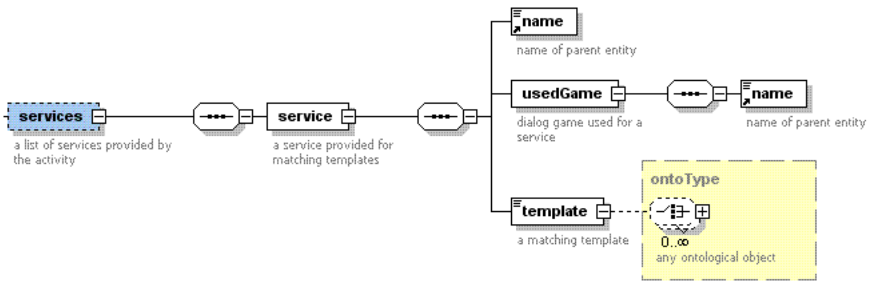


Figure A.2: XSD structure of *service* elements

A plan can have any number of *service* elements (see figure A.2). They describe services provided by the plan, and specify which game (as defined in the *games* section of the *Lisa* document) implements the service. The basic form of a *service* element is

```
service
  name: <string>
  usedGame
    name: <string>
  template
    <some ontological structure>
```

The template is used to match an input structure using the best match strategy. If the template is deemed a best match, the associated *usedGame* is executed.

A.2 Slot Definitions

Plans and bodies of code can have any number of slot definitions. These essentially define local variables with a *name* and a *type*.

A slot definition (see figure A.3) looks like this:

```
slot
  name: <string>
  type: <string>
  (
  value ... |
  slotValue ... |
  bindingValue ... |
  getABoxObject ... |
  string: <string>
  )
```

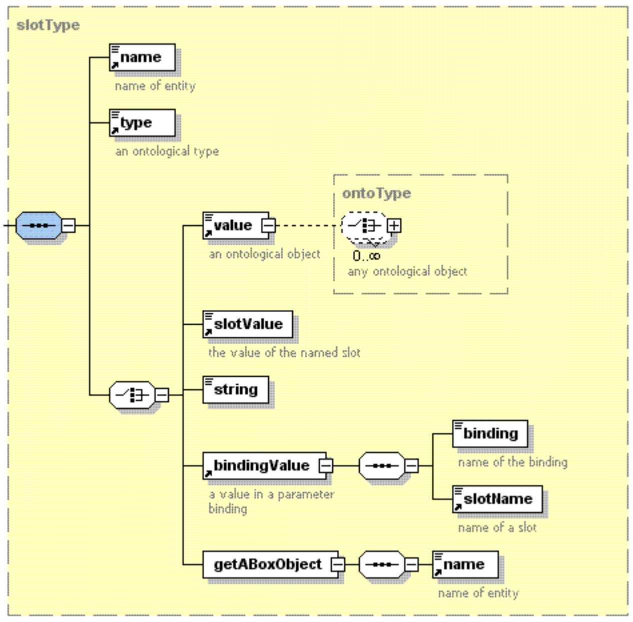


Figure A.3: XSD structure of a slot value specification

A.3 Value Specifications

A.3.1 Slots

The possibilities to assign a value are as follows (we refer to value specifications by the `<value>` tag in the following sections).

- *value*: this directly gives an ontological object to be bound to the slot
- *slotValue*: the slot *someName* is set to the value of another slot. *someName* can also be assigned a sub-value of the other slot. In this case, the sub-value is determined by specifying a *slot path* after the name of the second slot. Example:

  ```
  slot
    name: a
    type: t
    slotValue: b:has_x:has_y
  ```

  assigns the value of the role *has_y* of the object in the role *has_x* of the slot b to a.
- *bindingValue*: The source of the value in this case comes from another named binding. After a sub-game g is completed, its binding is available to the parent process under the name g.

  ```
  slot
    name: a
  ```

```
    type: t
    bindingValue
      binding g
      slotName b
```

assigns the value of slot b in binding g to slot a.

- *getABoxObject*:

```
  slot
    name: a
```

  retrieves a named object from the ontology that has previously been stored with the *assertObject* statement. If no such element exists, an exception is raised.

- *string*:

```
  slot
    name: a
    string: someString
```

  Sets a to the string value *someString*.

A.3.2 Relations

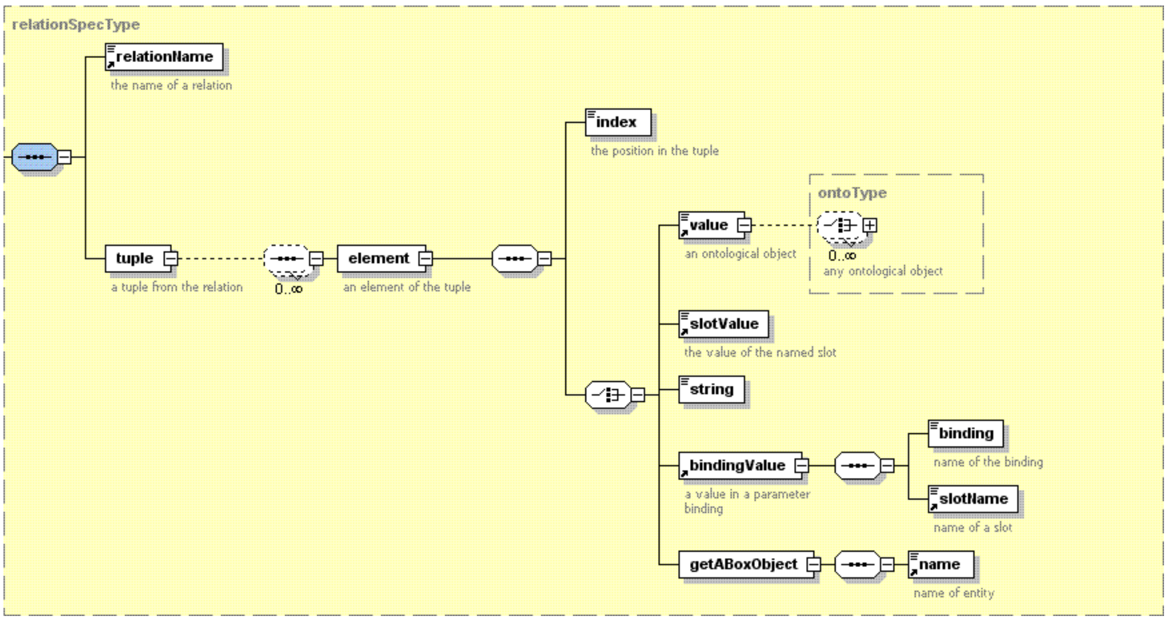


Figure A.4: XSD structure of relation value specifications

A.3.3 Conditions

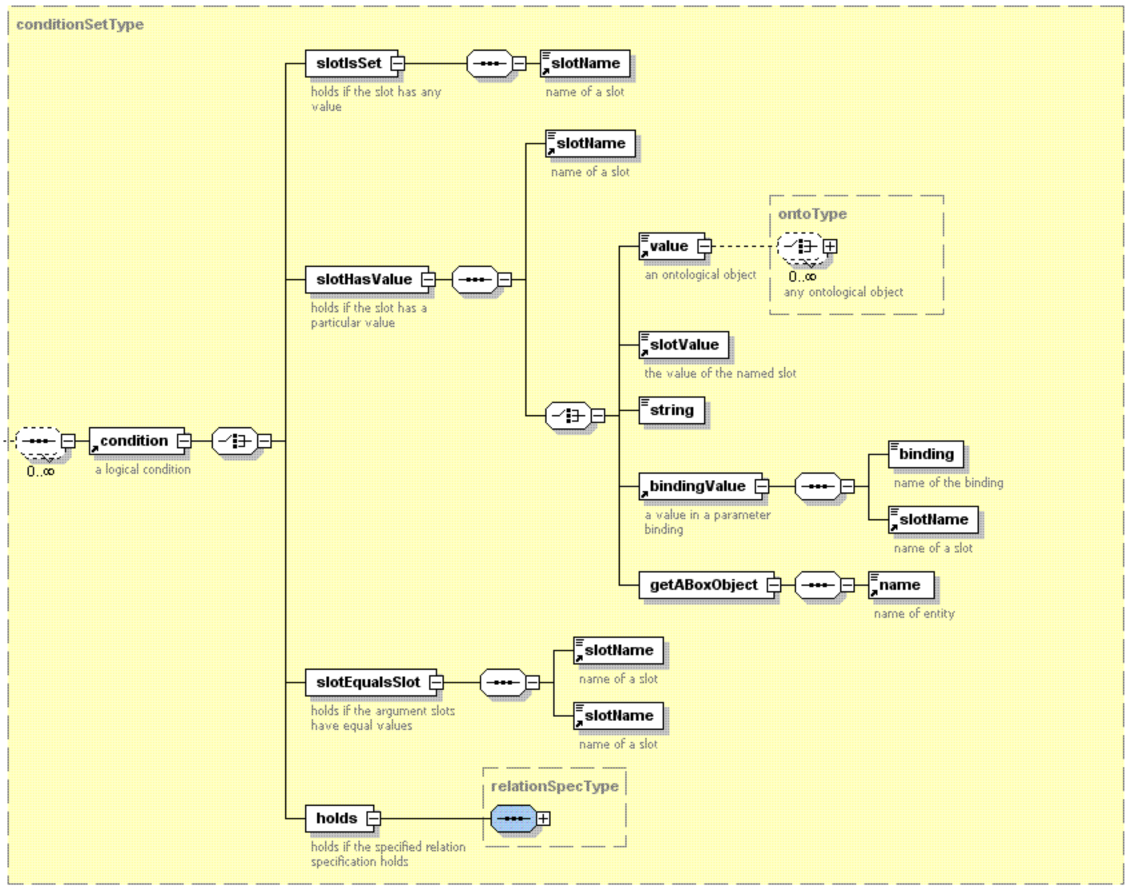


Figure A.5: XSD structure of a condition type

Condition elements (see figure A.5) are used to check for boolean conditions.

- *holds*: true if a relation r stored in the ontology holds for a particular n-tuple t of ontological objects. The tuple t is made up of ordered elements t_i ($i \in [0, n]$). In the condition element, the tuple elements are given with their indices and values.[2]

```
holds
  relationName: r
  tuple
    (element
      index: t_i
      <value>
    )*
```

- *slotHasValue*: true if a slot s has a particular value in the current binding.

[2]All tuple elements must be present in the condition. If the arity of the relation is different from the arity in the condition, an error is thrown.

```
slotHasValue
  slotName: s
  <value>
```

- *slotEqualsSlot*: true if both given slots have structurally equal values

```
slotEqualsSlot
  slotName: s1
  slotName: s2
```

- *slotIsSet*: true if the slot has any nonempty value

```
slotIsSet
  slotName: s
```

A.4 Body Elements

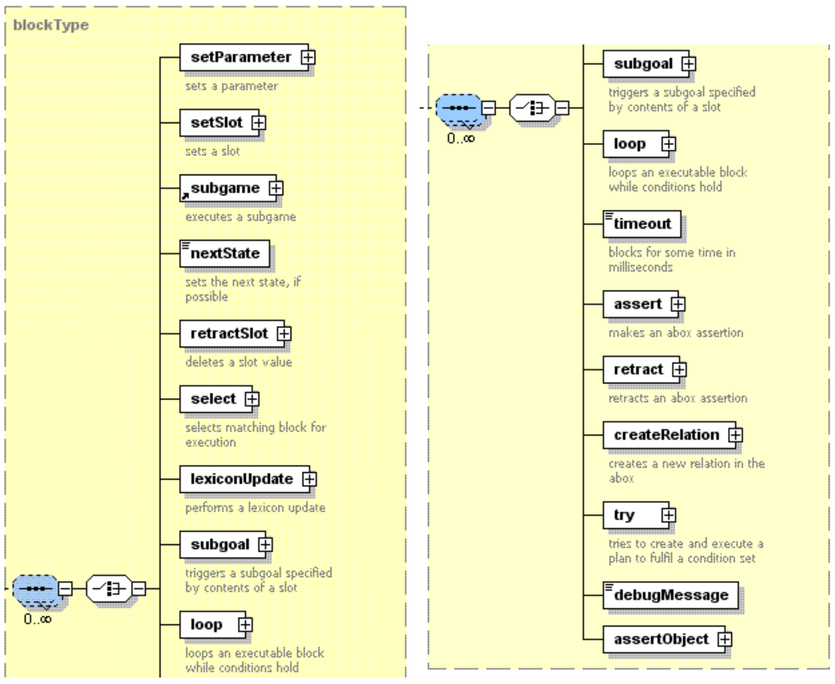

blockType
setParameter
sets a parameter
setSlot
sets a slot
subgame
executes a subgame
nextState
sets the next state, if possible
retractSlot
deletes a slot value
select
selects matching block for execution
lexiconUpdate
performs a lexicon update
subgoal
triggers a subgoal specified by contents of a slot
loop
loops an executable block while conditions hold
0..∞
subgoal
triggers a subgoal specified by contents of a slot
loop
loops an executable block while conditions hold
timeout
blocks for some time in milliseconds
assert
makes an abox assertion
retract
retracts an abox assertion
createRelation
creates a new relation in the abox
try
tries to create and execute a plan to fulfil a condition set
debugMessage
assertObject
0..∞

Figure A.6: XSD structure showing the possible statements in a *body*

- A *body* element (Figure A.6) holds a sequence of statements to execute.

A.4.1 General Statements

- *debugMessage* outputs a message. This is useful while developing plans.

```
debugMessage: someMessage
```

The example prints the text *"[Lisa] someMessage"* on the debug output.

- *body* inserts a sub-block of statements into another block.

  ```
  body
    (statement)*
  ```

- *nextState* (only valid while executing a game).

  ```
  nextState: stateName
  ```

  Specifies that outgoing edges are preferred that end in the state with the specified name.

- *subgame* executes a subgame.

  ```
  subgame
    name: someName
    gameType: someType
    (parameter
       name: paramName
       <value>
    )*
  ```

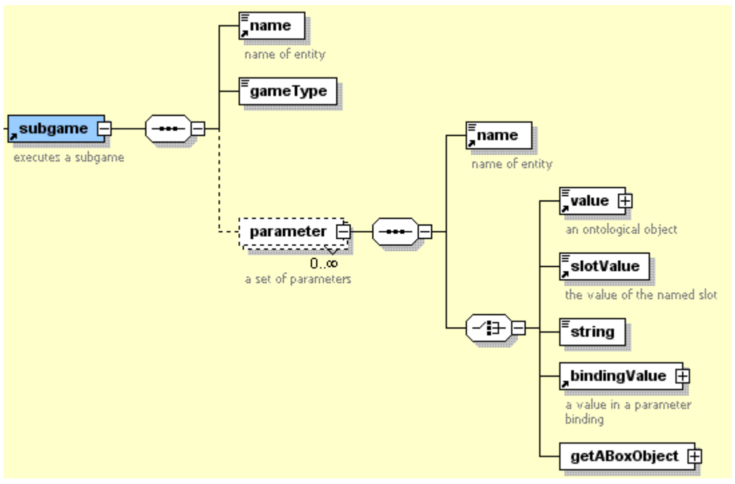


Figure A.7: XSD structure of *subgame* statement

A subgame of type *someType* is created and executed embedded in the current block (called the "parent" of the subgame). The block resumes execution after the game has finished. The parameter binding of the subgame is initialized with the values given by the *parameter* arguments. After the subgame has finished, the values in its binding is available in the parent's binding under its name *someName*.

- *timeout* suspends execution of the block for some time, and then resumes. Other processes running concurrently are not affected.

  ```
  timeout: timeoutValue
  ```

  TimeoutValue must be parseable as an integer value. The execution is suspended for *timeoutValue* milliseconds.

- *lexiconUpdate* updates the speech recognizer's lexicon.

  ```
  lexiconUpdate
    slotName: s
    path: a
    category: ADJ
  ```

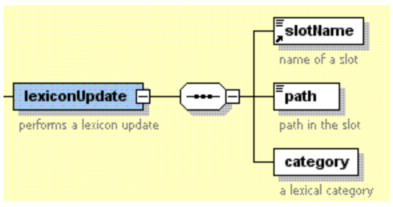


Figure A.8: XSD structure of *lexiconUpdate*

  The example asserts that the string found under the slot a of the slot s should be added to the lexicon as being in the adjective (ADJ) category.

- *subgoal* starts a subgoal.

  ```
  subgoal
    slotName: s
    [blocking]
  ```

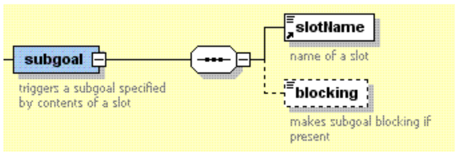

subgoal
triggers a subgoal specified by contents of a slot
slotName
name of a slot
blocking
makes subgoal blocking if present

Figure A.9: XSD structure of *subgoal*

  The statement tries to find and start a process p that provides a service described by the value of slot s (in effect, the same action that would happen if the user made the utterance in s). If the *blocking* element is present, the current process waits until p is finished, otherwise, it is executed in parallel.

- *try* invoke a planner.

```
try
  <condition_1>
  ...
  <condition_n>
```

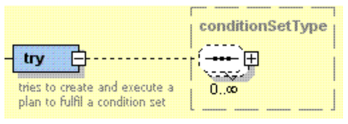

try
tries to create and execute a plan to fulfil a condition set
conditionSetType
0..∞

Figure A.10: XSD structure of *try*

invokes the external planner to generate and execute a plan to bring about the conditions, if possible.

A.4.2 Assertions and Relations

- *setSlot* sets a slot to some value.

```
setSlot
  slotName: someName
  <value>
```

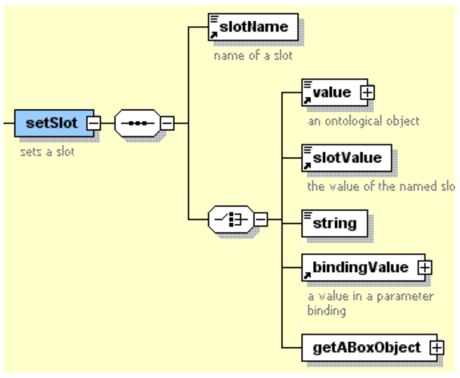


Figure A.11: XSD structure of *setSlot*

(see *Value Specifications* (Section A.3) for the possible forms of `<value>`)

- *retractSlot* retracts the value of a slot.

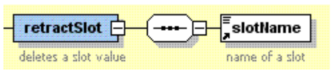


Figure A.12: XSD structure of *retractSlot*

```
retractSlot
  slotName: s
```

- *assertObject* (see figure A.13) dynamically stores an ontological named object in the CDE's ontology.

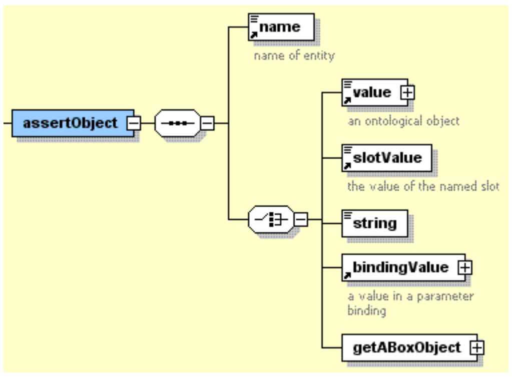


Figure A.13: XSD structure of *assertObject*

```
assertObject
  name: someName
  <value>
```

The result of evaluating `<value>` is stored in the ontology under the name *someName*. If there already is an object in the ontology with the same name, it is removed first. This is the method of choice for sharing object values with other processes in the same CDE, or to store them after the process has finished.

- *setParameter* sets parameters for outgoing edges of the current state.

```
setParameter
  name: someName
  <value>
```

If a parameter x is set, the role *has_x* of the dialogue act associated with an outgoing edge will be set to its value.

- *assert* adds a tuple of ontological objects to a relation in the ontology. If the relation does not exist, it is created.

```
assert
  relationName: someName
  tuple
    (element
      index: t_i
      <value>
    )*
```

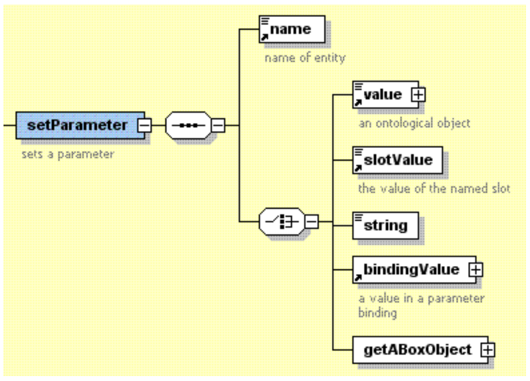


Figure A.14: XSD structure of *setParameter*

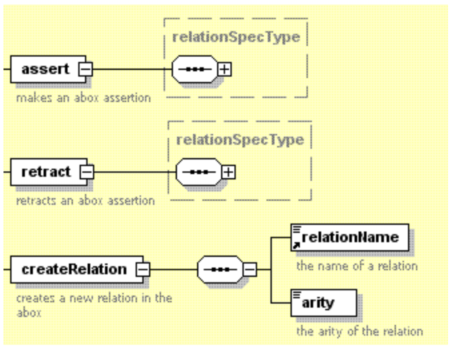


Figure A.15: XSD structure of relation operations

(for the meaning of the syntax of tuple specifications, see the corresponding entry for condition elements).

- *createRelation* creates a relation in the ontology.

```
createRelation
  relationName name
  arity: n
```

Creates a new relation *name* with arity n in the ontology that contains no tuples. It is an error to attempt to create a relation that already exists. When using the *assert* statement to add a tuple t to a relation that does not exist in the ontology, the relation is automatically created using the arity of t.

A.4.3 Control Structures

- *select* is a conditional statement (see figure A.16). The body of the first template argument whose conditions are fulfilled is executed.

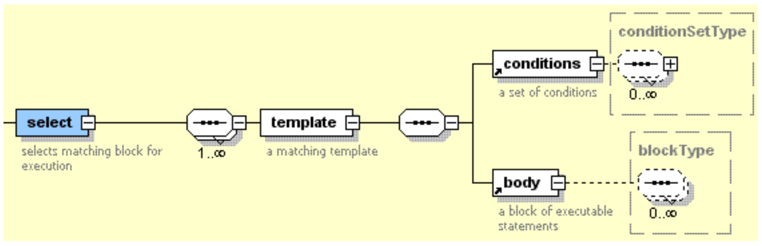


Figure A.16: XSD structure of *select*

```
select
  (template
    conditions
      (condition
        <condition>
      )*
    body
  )*
```

- *loop* defines a conditional execution loop.

```
loop
  [conditions
    <condition>*] // block-initial conditions
  body
    <body>
  [conditions
    <condition>*] // block-final conditions
```

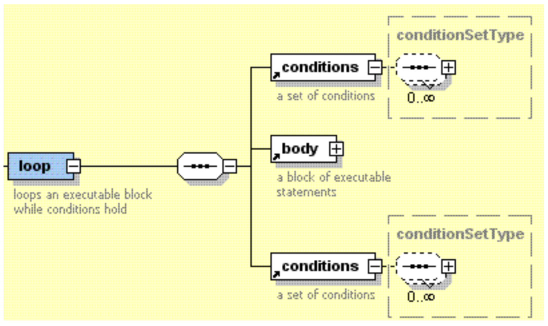


Figure A.17: XSD structure of a conditional execution *loop*

Executes a body of statements in a loop. The construct allows loops of the *do-while* and *repeat-until* flavors (or both combined). The body is executed repeatedly; for each iteration, the *block-initial-conditions* are checked before entering the body and the *block-final-conditions* after leaving the body. If a check fails, the iteration is terminated and execution continues after the loop.

A.5 Game Elements

Game elements define how a game type is implemented in the given process (see Figure A.18). If a game type is used by a process, there must be an element *game* that defines its implementation, and gives it a name.

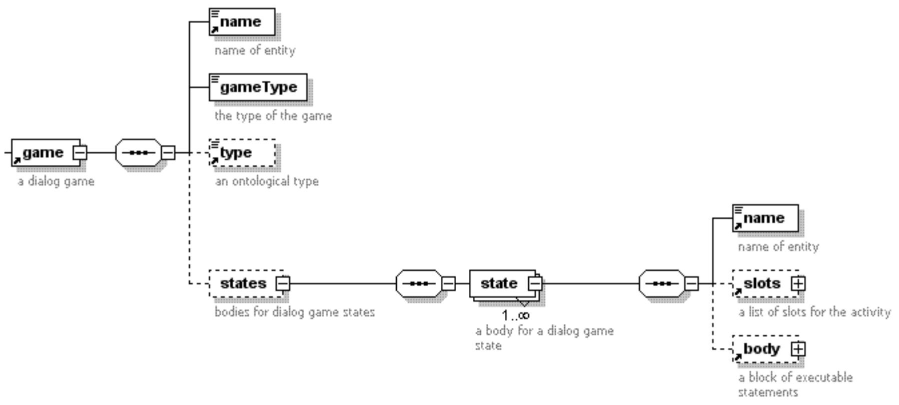


Figure A.18: XSD structure of a *game* type definition

```
game
  name: someName
  gameType: someTypeName
  states
    state
      name: stateName_n
      body
        [...]
    [...]
```

The example snippet redefines the game type *someTypeName* (from the game type definition) and gives it the name *someName* to be used by the plan the snippet occurs in. It then proceeds to define the bodies of states from this game type. In this case, the state *stateName_n* (which must occur in the game type) gets a new body definition. A plan can have arbitrarily many *game* elements. It is not necessary to redefine all (or even any) of the states from the game type; if a redefinition is missing, the respective body will just be treated as empty.

Appendix B

Elements of the *JenaLite* API

B.1 Concept

The *JenaLite* API is designed to provide a subset of the possibilities offered by the *Jena* RDF API.[1] Its purpose is to gain speed and memory efficiency at the expense of some aspects of *Jena*'s expressivity and inferential power that are not crucial for systems described in this thesis. The API defines a set of interfaces that can then be implemented by corresponding Java classes, for our framework, this was done by Mehdi Moniri.[2] The implementation allows to access and manipulate structures in RDF, the proprietary data format of the *Protégé* ontology editor, or the XML representation defined in Chapter 4.

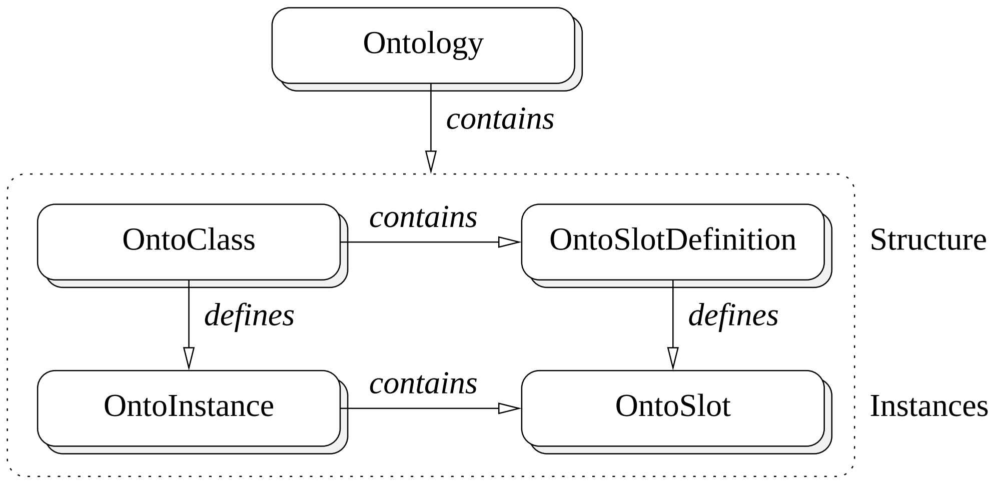


Figure B.1: *JenaLite* object hierarchy

Figure B.1 shows the hierarchy of objects using the *JenaLite* interfaces. An object implementing the *Ontology* interface represents an entire ontology. The structural knowledge is

[1]Project homepage: *http://jena.sourceforge.net*

[2]Programmatically, the interfaces are realized by abstract Java classes instead of Java *interface* constructs, because they already include base method implementations.

encoded in *OntoClass* and *OntoSlotDefinition* objects that specifiy, e. g., the inheritance relations, occurrence and requiredness of slots, and slot value types. Actual ontological instances and their attributes correspondingly are stored in *OntoInstance* and *OntoSlot* objects. Each of these object types offers a set of methods that allows to add, access and modify the ontology's content. The *JenaLite* interfaces were designed to match the requirements of actual use in the CDE framework, and incrementally extended during the implementation in some cases where the need for additional features was encountered.

B.2 Interfaces

JenaLite offers a view of ontological objects in terms of TFS and, correspondingly, XML objects that represent these TFS. Sub-objects can be referenced using *XPath* expressions that are similar to file system paths[3]; this mode of addressing is supported directly by the *JDOM* library used in the implementation of the interface. For example, the set of values of all top-level *has_owner* slots of an ontological instance is referenced by the expression *"/has_owner/*"*.

The interfaces were—on purpose—*not* designed to be minimal or orthogonal. That is, they exhibit more than minimal set of methods necessary to achieve their functionality. They instead feature additional convenience methods in cases where frequently occurring special uses of certain methods, or composite uses, were observed. For example, there are separate methods *getSlotInstanceValues()* and *getSlotInstanceValue()* in the *OntoInstance* interface, where the latter is only intended for situations where the programmer is sure there will be at most one single value returned. In this case, the extra step to extract the single element of the set returned by the latter method can be left out. If the programmer's assumption is wrong in these cases, an exception is thrown. Descriptions of the signatures of the five interfaces follow:

- *Ontology* (Figure B.2)[4]

  This interface allows operations on a whole dynamic ontology. Ontology data can be loaded with the *importData* method from files stored in the *Protégé*, XML, or RDF formats[5]. Instances can be added or removed dynamically, subclass relations can be tested, and best matches for instances over the entire ontology be found. There also is some support for namespaces and namespace abbreviations.

- *OntoClass* (Figure B.3)

  This interface gives access to the concepts of the ontology, including the instances and their slot definitions. As with the *Ontology* interface, instances can be added or removed. The *getSuperclassNames()* method only returns the set of names of the direct superclasses.

- *OntoSlotDefinition* (Figure B.4)

[3]see *http://www.w3.org/TR/xpath*

[4]For brevity reasons, this figure, and the following ones, use abbreviations for arguments and return values, e. g. *OntoClass* → *class*. Java already has a keyword *class*, so this name could not be used in the implementation.

[5]There are some incompatibilities between the file formats of *Protégé* versions. *JenaLite* supports the file format used in *Protégé* 3.0 beta, which was current during the *VirtualHuman* project.

method	**returns**	**remarks**
o.addInstance(*instance*)	—	
o.getInstance(*id*)	*instance*	
o.getNamespaceAbbreviations()	map(*string*→*string*)	Returns a mapping of abbreviation strings to namespaces
o.getOntoClass(*name*)	*ontoClass*	
o.getOntoClasses()	set(*ontoClass*)	
o.getStandardNamespace()	*string*	see *setStandardNamespace*
o.importData(*file*)	—	Automatically detects the data format and converts the contents to XML format if necessary
o.isSubclassOf(*cName1, cName2*)	*boolean*	
o.removeInstance(*instance*)	—	
o.setStandardNamespace(*namespace*)	—	One namespace string can be set as default; instances in this namespace can be referenced without qualification
o.someBestMatch(*instance*)	*instance*	Returns one best match for *instance* over all instances in the ontology
o.toXML()	*string*	

Figure B.2: The *Ontology* interface

The intended use of this interface is to get information about how slots are defined. It does not provide methods to create new slot definitions at run-time, since the taxonomic structure is static.[6]

- *OntoInstance* (Figure B.5)

  This is the most comprehensive interface in *JenaLite*. It provides methods to manipulate dynamic instances (*addSlot*, *replaceSlot*, *unifyWith* etc.), select and access parts of their information (*pathInstance*, *pathString*, etc.), and to test attributes (*isDynamic*, *hasSlotNamed*, etc.). The *path* arguments in the accessor methods *pathInstance* and *pathString*, the *add-/replace-/removeSlotAtPath* methods and *deepCopy* use the *XPath* notation to select substructures in the XML representation of the instance. The *Binding* defined by an instance is a data structure used by the framework that maps the top-level slot names to their values.

- *OntoSlot* (Figure B.6)

  This interface allows access and manipulation of slot values. These are of type *instance* or *string*, corresponding to complex resp. atomic values. For the purposes of the framework, it was not necessary to use the different types for atomic values available in *Protégé* (*string*, *integer*, *float*, *boolean* etc); instead, these values were converted to a textual (string) representation.

[6]In Mehdi Moniri's implementation of *JenaLite*, the slot definitions are created when an ontology is loaded. The class implementing the *Ontology* interface uses implementation-dependent methods of the class implementing *OntoSlotDefinition*.

method	**returns**	**remarks**
c.addInstance(*instance*)	—	
c.getDirectSlotDefinitions()	set*slotDefinition*	Returns the set of slots defined directly in the class, i. e., without slots inherited from superclasses
c.getInstance(*id*)	*instance*	
c.getInstances()	set(*instance*)	
c.getName()	*string*	
c.getOntology()	*ontology*	
c.getSlotDefinition(*slotName*)	*slotDefinition*	
c.getSlotDefinitions()	set(*slotDefinition*)	
c.getSlotNames()	set(*string*)	
c.getSuperclassNames()	set(*string*)	Gets the names of all *direct* superclasses only
c.removeInstance(*instance*)	—	
c.toXML()	*string*	

Figure B.3: The *OntoClass* interface

method	**returns**	**remarks**
d.getAllowedClasses()	set(*string*)	Returns the *names* of classes whose instances are allowed as slot values
d.getName()	*string*	
d.getValueType()	*string*	Can be *"instance"* or *"string"*
d.isMultiple()	*boolean*	
d.isRequired()	*boolean*	
d.toXML()	*string*	

Figure B.4: The *OntoSlotDefinition* interface

method	**returns**	**remarks**
i.addSlot(*slot*)	—	
i.addSlot(*slotName, instance*)	—	
i.addSlot(*slotName, valueString*)	—	
i.addSlotAtPath(*path, object*)	—	
i.addSlots(*binding*)	—	
i.containsSlot(*slotName*)	*boolean*	
i.deepCopy(*path, object*)	*instance*	
i.filterStrictlyExtends(set(*instance*))	set(*instance*)	filters the argument set and returns all elements that do not strictly extend i (cf. Section 4.4.2)
i.getAllUnifiableInstances()	set(*instance*)	
i.getAllUnifiableInstances(*isRestricted*)	set(*instance*)	
i.getBestMatches(set(*instance*))	set(*instance*)	
i.getBinding()	*binding*	
i.getBinding(*parentBinding*)	*binding*	
i.getId()	*string*	
i.getOntoClass()	*ontoClass*	
i.getOntoClassName()	*string*	
i.getOntology()	*ontology*	
i.getSlot(*name*)	*slot*	
i.getSlotInstanceValue(*slotName*))	*instance*	
i.getSlotInstanceValues(*slotName*))	set(*instance*)	
i.getSlots()	set(*slot*)	
i.getSlots(*name*)	set(*slot*)	
i.getSlotStringValue(*name*)	*string*	
i.getSlotStringValues(*name*)	set(*string*)	
i.getStaticUnifiableInstances(set(*instance*))	set(*instance*)	
i.getUnifiableInstances(set(*instance*))	set(*instance*)	
i.hasSlotNamed(*name*)	*boolean*	
i.isDynamic()	*boolean*	
i.isStrictlyExtendedBy(*instance*)	*boolean*	
i.isUnifiableWith(*instance*)	—	
i.pathInstance(*path*)	*instance*	
i.pathString(*path*)	*string*	
i.removeSlot(*slot*)	—	
i.removeSlotAtPath(*path*)	—	
i.replaceSlotAtPath(*path, valueObject*)	—	
i.setDynamic(*flag*)	—	sets the *dynamic* flag. A dynamic instance can be manipulated a run-time
i.setId(*id*)	—	
i.toElement()	*element*	
i.toRDF()	*string*	
i.toString()	*string*	
i.unifyWith(*instance*)	*instance*	
i.unifyWith(*instance, isRestricted*)	*instance*	

Figure B.5: The *OntoInstance* interface

method	**returns**	**remarks**
s.getInstanceValue()	*instance*	
s.getName()	*string*	
s.getOntoClass()	*ontoClass*	
s.getSlotDefinition()	*slotDefinition*	
s.getStringValue()	*string*	
s.getValue()	*object*	
s.getValueType()	*type*	Type can be one of "instance" or "string"
s.setInstance(*instance*)	—	
s.setStringValue(*string*)	—	
s.toRDF()	*string*	
s.toString()	*string*	

Figure B.6: The *OntoSlot* interface

Appendix C

Schemata for DSD and *directionML* documents

C.1 The Dialogue System Definition

Figure C.1 shows the schema for a dialogue system definition (DSD). A DSD consists of three parts: a *controller* section, a *participant* section and the *init* section.

- The controller section contains information necessary for the operation of the CDE controller, including the name of the dialogue system, the location of the system ontology file, the path to the subdirectory where application-specific data is located (*subdir*), the path to the game types definitions (*gameTypes*) and the channel specifications.
- the participants section contains one data structure for each participant CDE in the system, giving its name, type, ontology file, and activity definition file.
- the init section contains an (optional) *Lisa* block of statements. If statements are present, they are executed in order when the dialogue system is initialized. The main purpose of this section is to define the set of CDEs that are to be activated at bootup of the system.

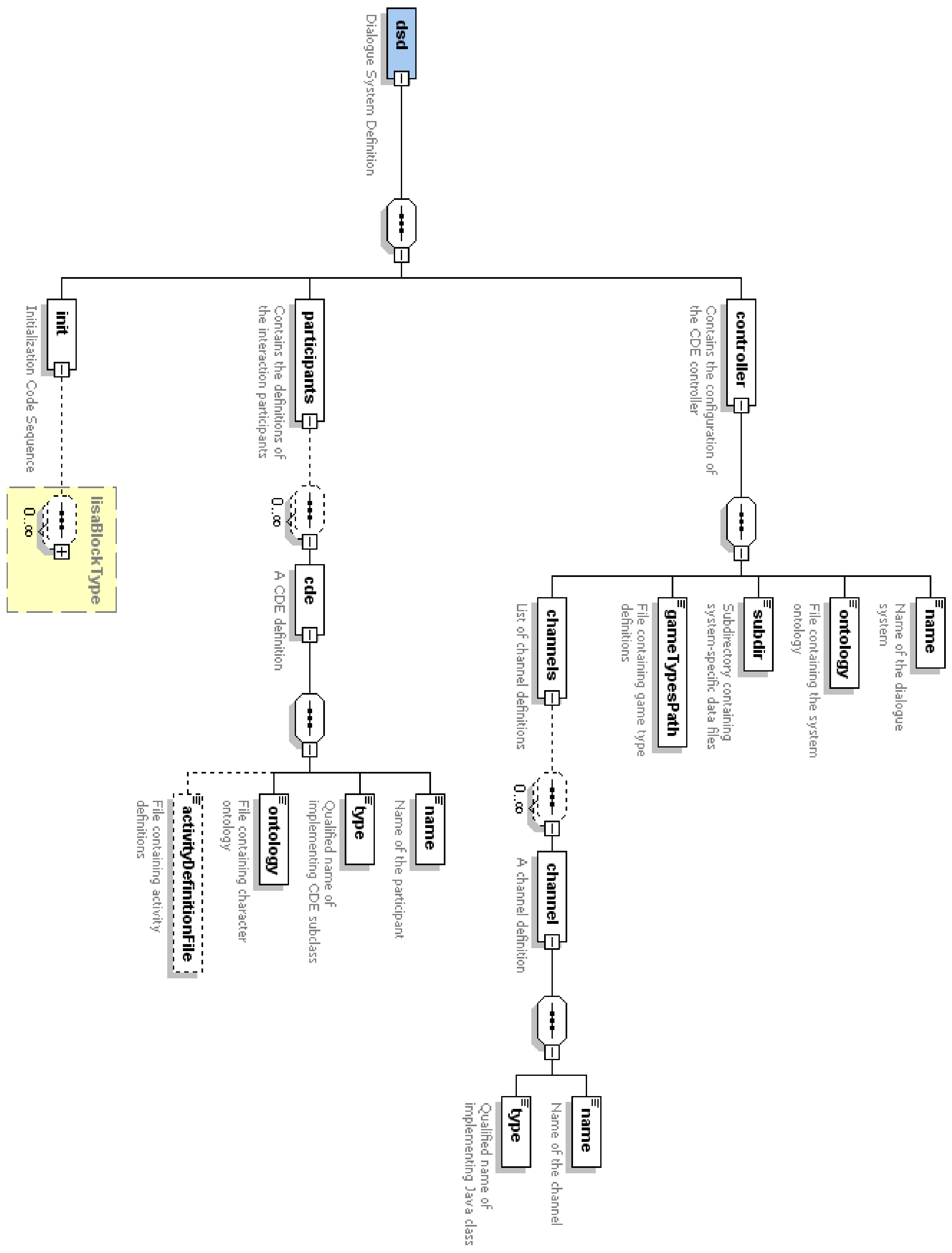

dsd
Dialogue System Definition
controller
Contains the configuration of the CDE controller
name
Name of the dialogue system
ontology
File containing the system ontology
subdir
Subdirectory containing system-specific data files
gameTypesPath
File containing game type definitions
channels
List of channel definitions
0..∞
channel
A channel definition
name
Name of the channel
type
Qualified name of implementing Java class
participants
Contains the definitions of the interaction participants
0..∞
cde
A CDE definition
name
Name of the participant
type
Qualified name of implementing CDE subclass
ontology
File containing character ontology
activityDefinitionFile
File containing activity definitions
init
Initialization Code Sequence
lisaBlockType
0..∞

Figure C.1: Schema for DSDs

C.2 The Direction Markup Language *directionML*

This section contains the schema definition of the *directionML* markup language introduced in *VirtualHuman*. It is used for both directions of the communication between the either the CDE framework or a set of individual CDEs, and a narration engine. Depending on sender and receiver, *directionML* messages serve different purposes; the main use is to set goals for CDEs and receive feedback about the execution and success state of the associated activities. The narration engine can also send meta-goals addressed at the framework to create or remove CDE instances from the scene. Via the *playerML* tag, instructions in playerML format can be issued that are routed to the player module without further intervention from the framework to allow for, e. g., story-controlled camera movements and events.

For reasons of space, the schematic definition of *directionML* messages is split in two parts: Figure C.2 shows the toplevel elements, and Figure C.3 shows the internal structure of a goal specification.

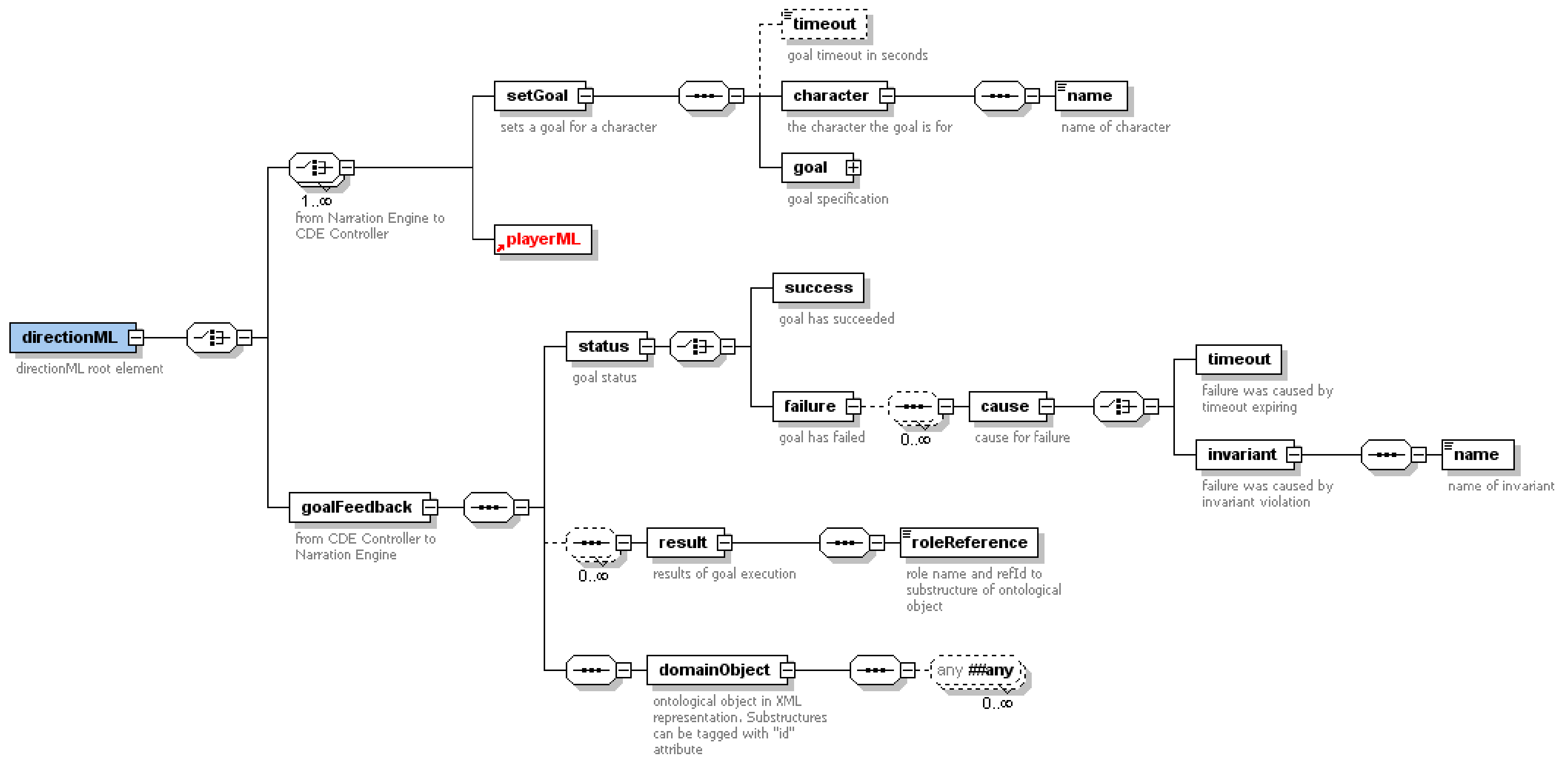

directionML
directionML root element
1..∞
from Narration Engine to CDE Controller
setGoal
sets a goal for a character
timeout
goal timeout in seconds
character
the character the goal is for
name
name of character
goal
goal specification
playerML
goalFeedback
from CDE Controller to Narration Engine
status
goal status
success
goal has succeeded
failure
goal has failed
0..∞
cause
cause for failure
timeout
failure was caused by timeout expiring
invariant
failure was caused by invariant violation
name
name of invariant
0..∞
result
results of goal execution
roleReference
role name and refId to substructure of ontological object
domainObject
ontological object in XML representation. Substructures can be tagged with "id" attribute
any ##any
0..∞

Figure C.2: Toplevel elements of *directionML*

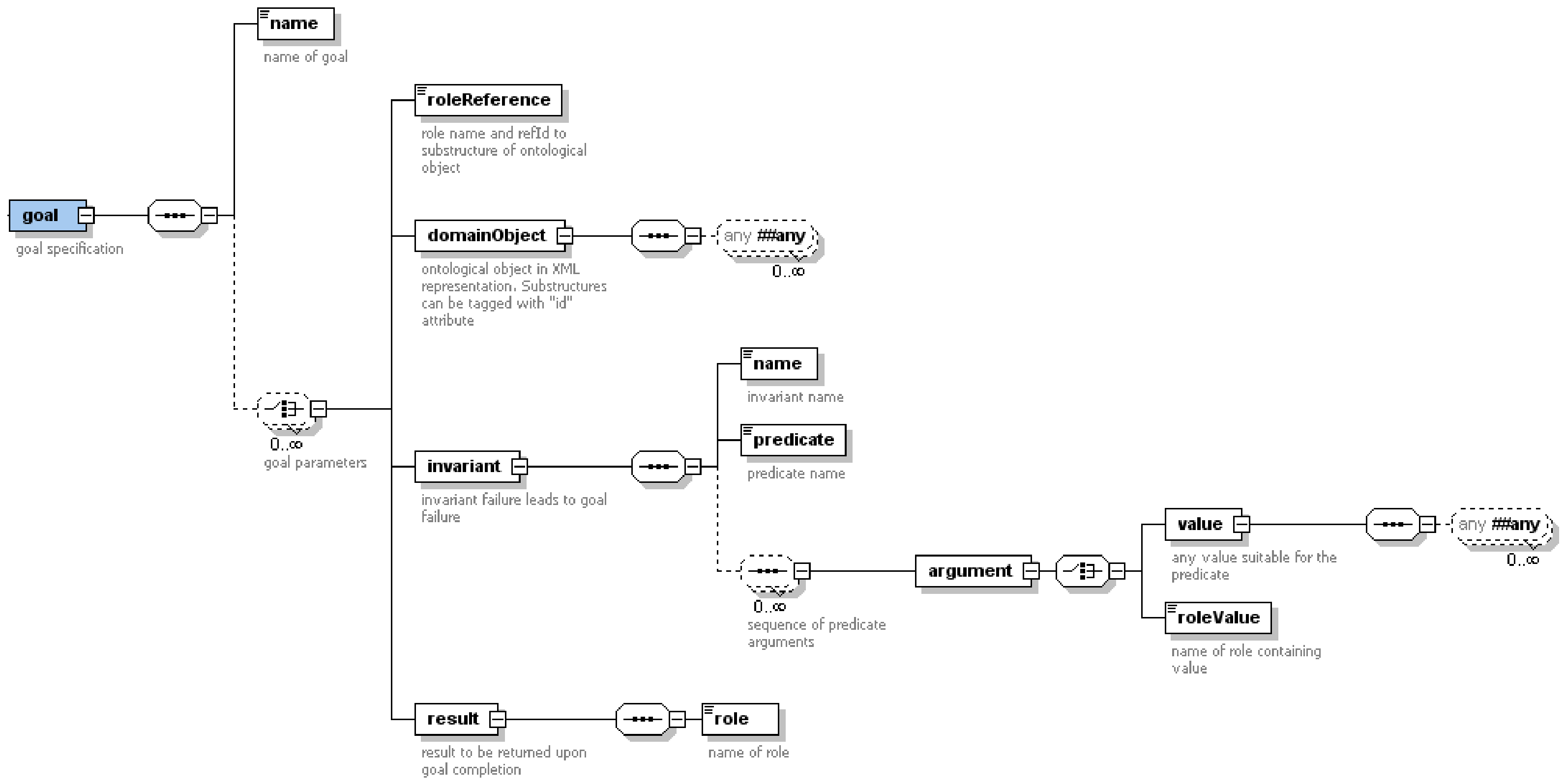

goal
goal specification
name
name of goal
0..∞
goal parameters
roleReference
role name and refId to substructure of ontological object
domainObject
ontological object in XML representation. Substructures can be tagged with "id" attribute
any ##any
0..∞
invariant
invariant failure leads to goal failure
name
invariant name
predicate
predicate name
0..∞
sequence of predicate arguments
argument
value
any value suitable for the predicate
any ##any
0..∞
roleValue
name of role containing value
result
result to be returned upon goal completion
role
name of role

Figure C.3: The *goal* specification in *directionML*

Appendix D

Communicative Acts

The model for conversational behavior generation is designed to make use of different ontologies to encode the knowledge bases of the characters. However, a minimum requirement is that there is a way to represent the foundational building blocks of the model, namely, activities, dialogue games, and dialogue acts; also, some auxiliary concepts are required if the application needs to make use of tuples, relations, and character models (which will almost always be the case).

These concepts, which are subconcepts of *RepresentationalObject* in the *VirtualHuman* ontology, have to be integrated with the base ontology of the system the framework should be adapted to. Figure 4.9 on page 110 and Figure 5.9 on page 140 depict parts of the subconcept tree below *RepresentationalObject*. In Figure D.1, the hierarchy of the communicative acts is shown; with the exception of *NonverbalAct* and *BackchannelFeedback* acts, which are used by *FADE* and some very application-specific dialogue acts, like the different types of “canned text” questions from *Clue*.

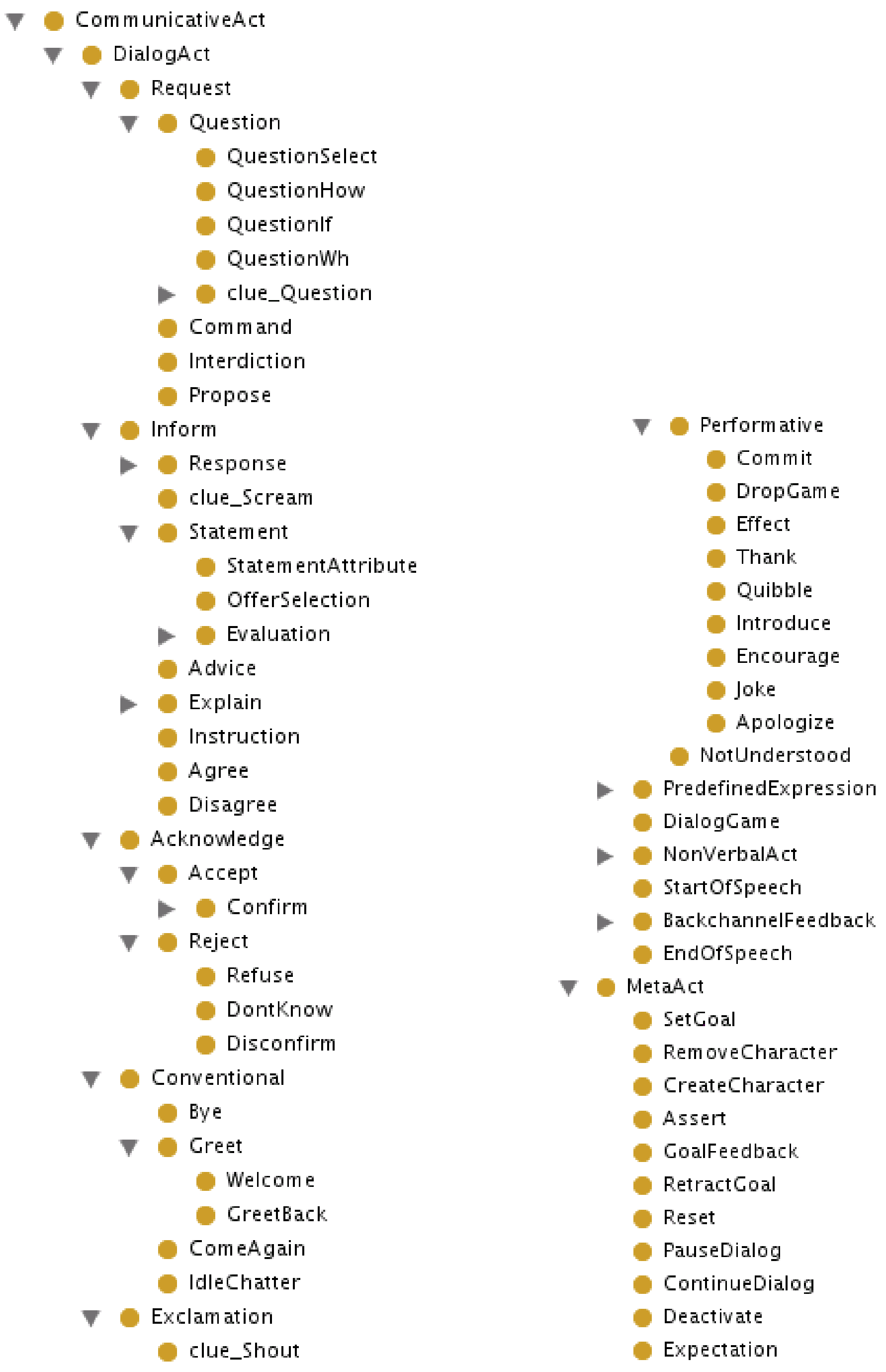


Figure D.1: Direct subcategories of *CommunicativeAct* in the ontology dialogue branch (the branch on the right side of the figure starts below *DialogueAct*)

Appendix E

Abbreviated XML Notation

In this document, we use an abbreviated notation for XML structures and the ontological objects represented in XML to avoid overly verbose examples. In the following, we informally explain the notation by way of an example. **Note:** ***This notation cannot be used to rewrite all well-formed XML documents unambiguously.*** For example, long strings wrapping into consecutive lines are a problem (which could be addressed by indenting them below the column, for example). It also cannot properly represent cases where elements have both textual and element children. However, this document does not contain any examples where these restrictions apply.

General XML. An example XML structure

```
<Alpha>
  <Beta attr1="aaa" attr2="bbb">
    <Gamma>
      hello, world!
    </Gamma>
    <Delta id="ccc">
      5.0
    </Delta>
    <Epsilon/>
  </Beta>
  <Zeta/>
</Epsilon>
```

is rewritten as

```
Alpha
  Beta attr1="aaa", attr2="bbb"
    Gamma: hello, world!
    Delta id="ccc": 5.0
    Epsilon
  Zeta
```

Basically, closing elements are omitted, the angle brackets marking elements are omitted, and the document structure of subelements is represented using indentation alone. A string child of an element is printed after a colon on the same line. To save additional space, an element that has only slots with atomic values may also be written with the slots and comma-separated values in parentheses, and leaving out attributes. The `Beta` element from above then would read `Beta(Gamma: hello world!, Delta: 5.0, Epsilon)`.

Ontological objects. Since we also use an XML representation for ontological objects (defined in section 4.3.4), the shorthand above also applies to them. Let *Alpha* be a class with roles *has_foo* and *has_bar* and *Beta* a class with one role *has_foo*. Let the role *has_foo* have a value type of *String*, and *has_bar* be a role with value type *Beta*. The following would be an XML notation for an instance of that class:

```
<Alpha>
  <has_foo>
    example string
  </has_foo>
  <has_bar>
    <Beta>
      <has_foo>
        another example string
      </has_foo>
    </Beta>
  </has_bar>
</Alpha>
```

or, in the shorthand notation,

```
Alpha
  has_foo: example string
  has_bar
    Beta
      has_foo: another example string
```

Bibliography

Russell L. Ackoff. From Data to Wisdom. *Journal of Applied Systems Analysis*, 16:34–39, 1989.

Jan Alexandersson. *Hybrid Discourse Modelling and Summarization for a Speech-to-Speech Translation System*. PhD thesis, University of the Saarland, Saarbrücken, Germany, 2003.

Jan Alexandersson. Some Ideas for the Automatic Aquisition of Dialogue Structure. In Anton Nijholt, Harry Bunt, Susann LuperFoy, Gert Veldhuijzen van Zanten, and Jan Schaake, editors, *Proceedings of the 11th Twente Workshop on Language Technology: Dialogue Management in Natural Language Dialogue Systems*, pages 149–158, University of Twente, Enschede, the Netherlands, 1996.

Jan Alexandersson and Tilman Becker. Overlay as the Basic Operation for Discourse Processing in a Multimodal Dialogue System. In *Proceedings of the IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 8–14, Seattle, WA, USA, 2001.

Jan Alexandersson and Paul Heisterkamp. Some Notes on the Complexity of Dialogues. In *Proceedings of the 1st SIGdial Workshop on Discourse and Dialogue*, pages 160–169, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

Jan Alexandersson, Elisabeth Maier, and Norbert Reithinger. A Robust and Efficient Three-layered Dialogue Component for a Speech-to-speech Translation System. In *Proceedings of the 7th Conference on European Chapter of the Association for Computational Linguistics*, pages 188–193, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

Jan Alexandersson, Tilman Becker, Ralf Engel, Markus Löckelt, Elsa Pecourt, Peter Poller, Norbert Pfleger, and Norbert Reithinger. Ends-based Dialogue Processing. In Robert Porzel, editor, *Proceedings of the 2nd Workshop on Scalable Natural Language Understanding (ScaNaLu)*, pages 25–32, Boston, MA, USA, 2004a.

Jan Alexandersson, Tilman Becker, and Norbert Pfleger. Scoring for Overlay based on Informational Distance. In *Proceedings of KONVENS 2004*, pages 1–4, Vienna, Austria, 2004b.

James Allen, Donna Byron, Dave Costello, Myrosia Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. TRIPS-911 System Demonstration. In *Proceedings of the NAACL/ANLP 2000 Workshop on Conversational Systems*, pages 33–35, Morristown, WA, USA, 2000a. Association for Computational Linguistics.

James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. An Architecture for a Generic Dialogue Shell. *Natural Language Engineering*, 6(3-4):213–228, 2000b.

James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum. The TRAINS Project: A Case Study in Building a Conversational Planning Agent. *Journal of Experimental and Theoretical AI (JETAI)*, 7:7–48, 1995.

James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. A Robust System for Natural Spoken Dialogue. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 62–70, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers.

James F. Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. Towards Conversational Human-Computer Interaction. *AI Magazine*, 22 (4):27–37, 2001a.

James F. Allen, George Ferguson, and Amanda Stent. An Architecture for More Realistic Conversational Systems. In *IUI '01: Proceedings of the 6th International Conference on Intelligent User Interfaces*, pages 1–8. ACM Press, 2001b.

Jens Allwood. An Activity Based Approach to Pragmatics. In Harry Bunt and Bill Black, editors, *Abduction, Belief and Context in Dialogue: Studies in Computational Pragmatics*, pages 47–80. John Benjamins, Amsterdam, the Netherlands, 2000.

Elisabeth André, Thomas Rist, and Jochen Müller. Integrating Reactive and Scripted Behaviors in a Life-like Presentation Agent. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents 1998)*, pages 261–268, New York, NY, USA, 1998. ACM Press.

Aristotle. *Topics*. Clarendon Press, Oxford, UK, 1928. W. D. Ross, editor.

John L. Austin. *How to do Things with Words*. Oxford University Press, London, UK, 1962.

Ruth S. Aylett. Narrative in Virtual Environments: Towards Emergent Narrative. In *Papers from the 1999 AAAI Fall Symposium, Technical report FS-99-01*, pages 83–86, Menlo Park, CA, USA, 1999. AAAI Press.

Ruth S. Aylett, Rui Figuieredo, Sandy Louchart, João Dias, and Ana Paiva. Making It Up as You Go Along - Improvising Stories for Pedagogical Purposes. In Jonathan Gratch, Michael Young, Ruth Aylett, Daniel Ballin, and Patrick Olivier, editors, *Proceedings of the 6th International Conference on Intelligent Virtual Agents (IVA)*, pages 307–315, Marina del Rey, CA, USA, 2006. Springer.

Kent Bach and Robert M. Harnish. *Linguistic Communication and Speech Acts*. Clerandon, Oxford, UK, 1962.

John A. Bateman. Upper Modeling: Organizing Knowledge for Natural Language Processing. In *Proceedings of the 5th International Workshop on Natural Language Generation*, pages 54–61, Dawson, PA, USA, 1990.

Sean Bechhofer, Carole Goble, and Ian Horrocks. Requirements of Ontology Languages. OntoWeb deliverable 4.1, University of Manchester, Manchester, United Kingdom, 2002.

Nicole Beringer, Daniela Oppermann, and Silke Steininger. Possible Lexical Indicators for Barge-In / Barge-Before in a Multimodal Man-Machine-Communication. SmartKom Technical Report 9, Ludwig-Maximilians-Universität (LMU), Munich, Germany, 2001.

Nicole Beringer, Ute Kartal, Katerina Louka, Florian Schiel, and Uli Türk. PROMISE - A Procedure for Multimodal Interactive System Evaluation. Technical Report 23, Ludwig-Maximilians-Universität (LMU), Munich, Germany, 2002.

Tim Berners-Lee, Dieter Fensel, James A. Hendler, Henry Lieberman, and Wolfgang Wahlster, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. The MIT Press, Cambridge, MA, USA, 2003.

Nate Blaylock. *Towards Tractable Agent-Based Dialogue*. PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, USA, 2005.

Dan Bohus and Alexander I. Rudnicky. RavenClaw: Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. In *Proceedings of EUROSPEECH-2003*, pages 597–600, Geneva, Switzerland, 2003.

Dan Bohus and Alexander I. Rudnicky. Error Handling in the RavenClaw Dialog Management Framework. In *HLT '05: Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 225–232, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

Johan Bos, Ewan Klein, Oliver Lemon, and Tetsushi Oka. DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In *Proceedings of the 4th SIGdial Workshop on Discourse and Dialogue*, pages 115–124, Sapporo, Japan, 2003.

Ronald J. Brachman and James G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

Michael E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, USA, 1987.

Jan Broersen, Mehdi Dastani, Joris Hulstijn, Zisheng Huang, and Leendert van der Torre. The BOID Architecture: Conflicts Between Beliefs, Obligations, Intentions and Desires. In *AGENTS '01: Proceedings of the 5th International Conference on Autonomous Agents*, pages 9–16. ACM Press, 2001.

Harry Bunt. A Framework for Dialogue Act Specification. In *Presented at the 4th ISO_SIGSEM Workshop on the Representation of Multimodal Semantic Information*, Tilburg, the Netherlands, 2005.
Available at *http://let.uvt.nl/general/people/bunt/docs/fdas.ps* (last access: 29.2.2008).

Harry Bunt. Context and Dialogue Control. *Think*, 3:19–31, 1994.

Sandra Carberry. Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.

Jean C. Carletta. Assessing Agreement on Classification Tasks: The Kappa Statistic. *Computational Linguistics*, 22(2):249–254, 1996.

Jean C. Carletta, Amy Isard, Stephen Isard, Jacqueline Kowtko, Gwyneth Doherty-Sneddon, and Anne Anderson. HCRC Dialogue Structure Coding Manual. Technical report, HCRC, Universities of Edinburgh and Glasgow, UK, 1996.

Jean C. Carletta, Stephen Isard, Gwyneth Doherty-Sneddon, Amy Isard, Jacqueline C. Kowtko, and Anne H. Anderson. The Reliability of a Dialogue Structure Coding Scheme. *Computational Linguistics*, 23(1):13–31, 1997.

Lauri Carlson. *Dialogue Games*. Synthese Language Library. D. Reidel Publishing Company, Dordrecht, the Netherlands, 1983.

Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, UK, 1992.

Justine Cassell. Embodied Conversational Agents: Representation and Intelligence in User Interface. *AI Magazine*, 22(3):67–83, 2001.

Justine Cassell, Hannes Vilhjálmsson, Kenny Chang, Timothy Bickmore, Lee Campbell, and Hao Yan. Requirements for an Architecture for Embodied Conversational Characters. In D. Thalmann and N. Thalmann, editors, *Computer Animation and Simulation '99*, Eurographics Series, pages 109–120. Springer, Vienna, Austria, 1999.

Marc Cavazza, Fred Charles, and Steven J. Mead. Interacting with Virtual Characters in Interactive Storytelling. In *Proceeedings of the Autonomous Agents Conference (AAMAS'02)*, pages 318–325, Bologna, Italy, 2002.

Chambers Editors. *The Chambers Dictionary*. Chambers Harrap Publishers, Cambridge, UK, 1993.

Herbert H. Clark. *Using Language*. Cambridge University Press, Cambridge, UK, 1996.

Herbert H. Clark and Edward F. Schaefer. Contributing to Discourse. *Cognitive Science*, 13 (2):259–294, 1989.

Philip R. Cohen. Discourse and Dialogue: Dialogue Modeling. In Ron Cole, editor, *Survey of the State of the Art in Human Language Technology*, pages 204–210. Cambridge University Press, New York, NY, USA, 1997.

Philip R. Cohen and Hector J. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42:213–261, 1990.

Philip R. Cohen, James A. Pittman, Ira Smith, and Tzu-Chieh Yang. QuickSet: A Multimodal Interface for Distributed Interactive Simulation. Abstract for a demo at the 9th Annual ACM Symposium on User-Interface Software and Technology, 1996.
Available at *http://citeseer.ist.psu.edu/cohen96quickset.html* (last accessed 12.07.2007).

Philip R. Cohen, Michael Johnston, David R. McGee, Sharon L. Oviatt, Jay A. Pittman, Ira Smith, Liang Chen, and Josh Clow. QuickSet: Multimodal Interaction for Distributed Applications. In *Proceedings of the 5th International Multimedia Conference*, pages 31–40, Seattle, WA, USA, 1997. ACM Press.

Philip R. Cohen, David R. McGee, and Josh Clow. The Efficiency of Multimodal Interaction for a Map-Based Task. In *Proceedings of the Applied Natural Language Processing Conference (ANLP'00)*, pages 331–338, Seattle, WA, USA, 2000. Morgan Kaufmann Publishers.

Philip R. Cohen, Rachel Coulston, and Kelly Krout. Multimodal Interaction During Multiparty Dialogues: Initial Results. In *4th IEEE International Conference on Multimodal Interfaces (ICMI)*, pages 448–453, Pittsburgh, PA, USA, 2002.

Mark G. Core and James F. Allen. Coding Dialogues with the DAMSL Annotation Scheme. In David R. Traum, editor, *Working Notes: AAAI Fall Symposium on Communicative Action in Humans and Machines*, pages 28–35, Menlo Park, CA, USA, 1997. American Association for Artificial Intelligence.

Chris Crawford. Assumptions Underlying the Erasmatron Interactive Storytelling Engine. In M. Mateas and P. Sengers, editors, *Proceedings of the AAAI Fall Symposium: Narrative Intelligence*, Technical Report, pages 112–114, Menlo Park, CA, USA, 1999. AAAI Press.

Nils Dahlbäck and Arne Jönsson. An Empirically Based Computationally Tractable Dialogue Model. In *Proceedings of the Fourteenth Annual Meeting of The Cognitive Science Society*, pages 785–790, Bloomington, IN, USA, 1992.

Jonn E. Deaton, Charles Barba, Tom Santarelli, Larry Rosenzweig, Vance Souders, Chris McCollum, Jason Seip, Bruce W. Knerr, and Michael J. Singer. Virtual Environment Cultural Training for Operational Readiness (VECTOR). *Virtual Reality*, 8(3):156–167, 2005.

Song Dongyi. *Combining Speech User Interfaces for Different Applications*. PhD thesis, Ludwig-Maximilians-Universität (LMU), Munich, Germany, 2006.

Dejing Dou, Drew V. McDermott, and Peishen Qi. Ontology Translation on the Semantic Web. In *Proceedings of the International Conference on Ontologies, Databases and Application of Semantics (ODBASE)*, pages 952–969, 2003.

Patrick Doyle. Believability Through Context Using "Knowledge in the World" to Create Intelligent Characters. In *AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 342–349, New York, NY, USA, 2002. ACM Press.

Laila Dybkjær and Niels Ole Bernsen. Usability Evaluation in Spoken Language Dialogue Systems. In *Proceedings of the Workshop on Evaluation for Language and Dialogue Systems*, pages 1–10, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

Karolina Eliasson. Integrating a Discourse Model with a Learning Case-Based Reasoning System. In Claire Gardent and Bertrand Gaiffe, editors, *9th Workshop on the Semantics and Pragmatics of Dialogue (DIALOR)*, pages 29–36, LORIA, Nancy, France, 2005.

Ralf Engel. SPIN: Language Understanding for Spoken Dialogue Systems Using a Production System Approach. In *Proceedings of the 7th International Conference on Spoken Language Processing*, pages 2717–2720, Denver, CO, USA, 2002.

George Ferguson and James F. Allen. TRIPS: An Integrated Intelligent Problem-Solving Assistant. In *AAAI '98/IAAI '98: Proceedings of the Fifteenth National/10th Conference on Artificial*

Intelligence/Innovative Applications of Artificial Intelligence, pages 567–572. American Association for Artificial Intelligence, 1998.

George M. Ferguson, James F. Allen, Brad W. Miller, and Eric K. Ringger. The Design and Implementation of the TRAINS-96 System: A Prototype Mixed-Initiative Planning Assistant. TRAINS Technical Note 96-5, Computer Science Dept., University of Rochester, Rochester, NY, USA, October 1996.

R. E. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

Tim Finin, Rich Fritzson, Don McKay, and Robin McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, NY, USA, 1990.

Annika Flycht-Eriksson. A Survey of Knowledge Sources in Dialogue Systems. *Electronic Transactions on Artificial Intelligence*, 3(D):5–32, 1999.

L. T. F. Gamut. *Logic, Language, and Meaning*, volume 2: Intensional Logic and Logical Grammar. University of Chicago Press, Chicago, IL, USA, 1991.

Patrick Gebhard. ALMA - A Layered Model of Affect. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 29–36, Utrecht, the Netherlands, 2005.

Patrick Gebhard. *Emotionalisierung interaktiver virtueller Charaktere - ein mehrschichtiges Computermodell zur Erzeugung und Simulation von Gefühlen in Echtzeit*. PhD thesis, University of the Saarland, Saarbrücken, Germany, 2007.

Patrick Gebhard, Michael Kipp, Martin Klesen, and Thomas Rist. Authoring Scenes for Adaptive, Interactive Performances. In *AAMAS '03: Proceedings of the 2nd international joint conference on Autonomous agents and multiagent systems*, pages 725–732, New York, NY, USA, 2003. ACM Press.

John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

Mike Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Mike Wooldridge. The Belief-Desire-Intention Model of Agency. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 1–10. Springer-Verlag: Heidelberg, Germany, 1999.

Alfonso Gerevini and Derek Long. Plan Constraints and Preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia, Italy, 2005.

Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Drew McDermott, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David Wilkins. PDDL—The Planning Domain Definition Language, 1998.

Ehsan Gholamsaghaee. Adapting JSHOP to a Dialog Framework with an Ontological Domain Description. Bachelor's thesis, Department of Computer Science, University of the Saarland, 2006.

Jonathan Ginzburg. Interrogatives: Questions, Facts, and Dialogue. In Shalom Lappin, editor, *The Handbook of Contemporary Semantic Theory*, pages 385–422. Blackwell Publishers, Oxford, UK, 1996.

Stefan Göbel, Oliver Schneider, Ido Iurgel, Axel Feix, Christian Knöpfle, and Alexander Rettig. Virtual Human: Storytelling and Computer Graphics for a Virtual Human Platform. In *Lecture Notes in Computer Science 3105 (TIDSE'04)*, pages 79–88, Darmstadt, Germany, 2004. Springer.

Stefan Göbel, Felicitas Becker, and Axel Feix. INSCAPE: Storymodels for Interactive Storytelling and Edutainment Applications. In Gérard Subsol, editor, *International Conference on Virtual Storytelling*, volume 3805 of *Lecture Notes in Computer Science*, pages 168–171. Springer, 2005.

Stefan Göbel, Ido Iurgel, Markus Rössler, Frank Hülsken, and Christian Eckes. Design and Narrative Structure for the Virtual Human Scenarios. *International Journal of Virtual Reality*, 6(4):1–10, 2007.

Paul Graham. *On Lisp: Advanced Techniques for Common Lisp*. Prentice Hall, London, UK, 1994.

Herbert P. Grice. *Syntax and Semantics*, volume 3: Speech Acts, P. Cole and J. Morgan (eds.), chapter "Logic and Conversation", pages 41–58. Academic Press, New York, NY, USA, 1975.

Derek Gross, James F. Allen, and David R. Traum. The TRAINS 91 Dialogues. Technical Report TN92-1, Computer Science Department, University of Rochester, NY, USA, 1993.

Barbara Grosz. Discourse and Dialogue: Overview. In Ron Cole, editor, *Survey of the State of the Art in Human Language Technology*, pages 199–201. Cambridge University Press, New York, NY, USA, 1997.

Barbara Grosz and Sarit Kraus. Collaborative Plans for Complex Group Action. *Artificial Intelligence*, 86(2):269–357, 1996.

Barbara Grosz and Sarit Kraus. *Foundations and Theories of Rational Agencies, A. Rao and M. Wooldridge, Eds.*, chapter The Evolution of SharedPlans, pages 227–262. Kluwer, Dordrecht, the Netherlands, 1999.

Barbara J. Grosz and Candace L. Sidner. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics*, 12(3):175–204, 1986.

Barbara J. Grosz and Candace L. Sidner. Plans for Discourse. In Philip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors, *Intentions in Communication*, chapter 20, pages 417–444. MIT Press, Cambridge, MA, USA, 1990.

Thomas R. Gruber. A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5 (2):199–220, 1993.

Nicola Guarino. Formal Ontology, Conceptual Analysis and Knowledge Representation. *International Journal on Human-Computer Studies, Special Issue on Formal Ontology, Conceptual Analysis and Knowledge Representation*, 43(5–6):625–640, 1995.

Iryna Gurevych, Robert Porzel, and Rainer Malaka. Modeling Domain Knowledge: Know-How and Know-What. In Wolfgang Wahlster, editor, *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 71–84. Springer Verlag, Berlin, Germany, 2006.

Thomas Heider and Thomas Kirste. Supporting Goal Based Interaction with Dynamic Intelligent Environments. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 596–600, Lyon, France, 2002.

Gerd Herzog, Heinz Kirchmann, Stefan Merten, Alassane Ndiaye, and Peter Poller. MULTIPLATFORM Testbed: An Integration Platform for Multimodal Dialog Systems. In H. Cunningham and J. Patrick, editors, *Proceedings of the HLT-NAACL 2003 Workshop on Software Engineering and Architecture of Language Technology Systems (SEALTS)*, pages 75–82, Edmonton, Canada, 2003.

Ari Hiltunen. *Aristotle in Hollywood. Visual Stories that Work*. Intellect Books, Bristol, UK, 2002.

Jean-Michel Hoc. *Cognitive Psychology of Planning*. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

Jürgen Hoffmeister, Christel Müller, and Engelbert Westkämper. *Sprachtechnologie in der Anwendung: Sprachportale*. Springer, Berlin, Germany, 2007.

Frank Hülsken, Christian Eckes, Roland Kuck, Jörg Unterberg, and Sophie Jörg. Modeling and Animating Virtual Humans for Real-Time Applications. *International Journal of Virtual Reality*, 6(4):11–20, 2007.

Joris Hulstijn. Dialogue Games are Recipes for Joint Action. In M. Poesio and D. Traum, editors, *Formal Semantics and Pragmatics of Dialogue (Gotalog'00)*, volume 00-5, pages 99–106, Gothenburg University, Gothenburg, Sweden, 2000a.

Joris Hulstijn. *Dialogue Models for Inquiry and Transaction*. PhD thesis, University of Twente, the Netherlands, 2000b.

Chung-Hee Hwang and Lenhart K. Schubert. Episodic Logic: A Comprehensive Natural Representation for Language Understanding. *Mind and Machine*, 3(4):381–419, 1993.

L. C. Jain, editor. *Fusing Intelligence in Virtual Agents*. Springer, 2008. (to appear).

Anthony Jameson, Angela Mahr, Michael Kruppa, Andreas Rieger, and Robert Schleicher. Looking for Unexpected Consequences of Interface Design Decisions: The MeMo Workbench. In *Proceedings of the 6th International Workshop on TAsk MOdels and DIAgrams (TAMODIA 2007)*, pages 279–286, Toulouse, France, 2007.

Oliver P. John and Sanjay Srivastava. The Big-Five Trait Taxonomy: History, Measurement, and Theoretical Perspectives. In L. A. Pervin and O. P. John, editors, *Handbook of Personality Theory and Research*, volume 2, pages 102–138. Guildford Press, New York, NY, USA, 1999.

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 Annual Meeting of the Association for Computational Linguistics*, pages 376–383, Philadelphia, PA, USA, 2001.

Arne Jönsson. A Model for Habitable and Efficient Dialogue Management for Natural Language Interaction. *Natural Language Engineering*, 3(2):103–122, 1997.

Arne Jönsson. *Dialogue Management for Natural Language Interfaces – An Empiricial Approach*. PhD thesis, Linköping Studies in Science and Technology, No 312, Linköping, Sweden, 1993.

Arne Jönsson and Nils Dahlbäck. Talking to a Computer is not Like Talking to Your Best Friend. In *Proceedings of The first Scandinavian Conference on Artificial Intelligence*, pages 53–68, Tromsø, Norway, 1988.

Yvonne Jung and Christian Knöpfle. Real Time Rendering and Animation of Virtual Characters. *International Journal of Virtual Reality*, 6(4):55–66, 2007.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice-Hall, Upper Saddle River, NJ, USA, 2000.

Marcelo Kallmann and Daniel Thalmann. Modeling Objects for Interaction Tasks. In *EGCAS'98, 9th Eurographics Workshop on Animation and Simulation*, pages 73–86, Lisbon, Portugal, 1998.

Benjamin Kempe, Norbert Pfleger, and Markus Löckelt. Generating Verbal and Nonverbal Utterances for Virtual Characters. In *Proceedings of the International Conference on Virtual Storytelling (ICVS) 2005*, pages 73–78, Strasbourg, France, 2005.

Michael Kipp, Kerstin H. Kipp, Alassane Ndiaye, and Patrick Gebhard. Evaluating the Tangible Interface and Virtual Characters in the Interactive COHIBIT Exhibit. In *Proceedings of the 6th International Conference on Intelligent Virtual Agents (IVA'06)*, pages 434–444, Marina Del Rey, CA, USA, 2006.

Martin Klesen and Patrick Gebhard. Affective Multimodal Control of Virtual Characters. *International Journal of Virtual Reality*, 6(4):43–54, 2007.

Martin Klesen, Michael Kipp, Patrick Gebhard, and Thomas Rist. Staging Exhibitions: Methods and Tools for Modelling Narrative Structure to Produce Interactive Performances with Virtual Actors. *Virtual Reality, Special Issue of Virtual Reality on Storytelling in Virtual Environments*, 7(1):17–29, 2003.

Christian Knöpfle and Yvonne Jung. The Virtual Human Platform: Simplifying the Use of Virtual Characters. *International Journal of Virtual Reality*, 5(2):25–30, 2006.

Stefan Kopp, Lars Gesellensetter, Nicole C. Krämer, and Ipke Wachsmuth. A Conversational Agent as Museum Guide - Design and Evaluation of a Real-World Application. In *Proceedings of the 5th International Conference on Intelligent Virtual Agents (IVA'05)*, pages 329–343, Kos, Greece, 2005.

Jacqueline Kowtko, Stephen Isard, and Gwyneth Doherty. Conversational Games within Dialogue. In M. Caenepeel, J. L. Delin, L. Oversteegen, G. Redeker, and J. Sanders, editors, *Proceedings of the DANDI Workshop on Discourse Coherence*, University of Edinburgh Centre for Cognitive Science, Edinburgh, UK, 1991.

Nicole C. Krämer and Gary Bente. Virtuelle Helfer: Embodied Conversational Agents in der Mensch-Computer-Interaktion. In G. Bente, N. C. Krmer, and A. Petersen, editors, *Virtuelle Realitäten*, pages 203–225. Hogref, Göttingen, Germany, 2002.

Ivana Kruijff-Korbayová, Elena Karagjiosova, Kepa J. Rodríguez, and Stina Ericsson. A Dialogue System with Contextually Appopriate Spoken Output Intonation. In *Proceedings of EACL-2003*, pages 199–202, Budapest, Hungary, 2003.

Ivana Kruijff-Korbayová, Gabriel Amores, Nate Blaylock, Stina Ericsson, Guillermo Pérez, Kalliroi Georgila, Michael Kaisser, Staffan Larsson, Oliver Lemon, Pilar Manchón, and Jan Schehl. Extended Information State Modeling. Technical report, TALK deliverable 3.1, 2006.

Michael Kruppa, Lübomira Spassova, and Michael Schmitz. The Virtual Room Inhabitant - Intuitive Interaction With Intelligent Environments. In S. Zhang and R. Jarvis, editors, *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence (AI05)*, pages 225–234, Sydney, Australia, 2005.

John E. Laird and Paul Rosenbloom. The Evolution of the Soar Cognitive Architecture. In David M. Steier and Tom M. Mitchell, editors, *Mind Matters: A Tribute to Allen Newell*, pages 1–50. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1996.

Lynn Lambert and Sandra Carberry. A Tripartite Plan-Based Model of Dialogue. In *Proceedings of the 29th annual meeting of the Association for Computational Linguistics*, pages 47–54, Morristown, NJ, USA, 1991.

Marion Langer, Frank Hülsken, and Christian Eckes. Evaluation des T24-Demonstrators im BMBF-Forschungsprojekt "Virtual Human". Technical Report 3, Fraunhofer-Institut für Medienkommunikation, St. Augustin, Germany, 2005.

Staffan Larsson. *Issue-Based Dialogue Management*. PhD thesis, Department of Linguistics, Gothenburg University, Gothenburg, Sweden, 2002.

Staffan Larsson and David R. Traum. Information State and Dialogue Management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3–4):323–340, 2000.

Staffan Larsson, Peter Ljunglöf, Robin Cooper, Elisabet Engdahl, and Stina Ericsson. GoDiS - An Accommodating Dialogue System. In *Proceedings of ANLP/NAACL-2000 Workshop on Conversational Systems*, pages 7–10, Seattle, WA, USA, 2000.

Oliver Lemon, Alexander Gruenstein, and Stanley Peters. Collaborative Activities and Multi-Tasking in Dialogue Systems – Towards Natural Dialogue with Robots. *Traitement automatique des langues: Special issue on dialogue*, 43(2):131–154, 2002.

David Lewis. Scorekeeping in a language game. *Journal of Philosophical Logic*, 8:339–359, 1979.

Anthony Liew. Understanding Data, Information, Knowledge and their Inter-Relationships. *Journal of Knowledge Management Practice*, 8(2), 2007.
Available at *http://www.tlainc.com/articl134.htm* (last accessed 08.02.07).

Craig A. Lindley. Story and Narrative Structures in Computer Games. In Brunhild Bushoff, editor, *Developing Interactive Narrative Content: sagas/sagasnet reader*. HighText, Munich, Germany, 2005.

Peter Ljunglöf, Björn Bringert, Robin Cooper, Ann-Charlotte Forslund, David Hjelm, Rebecca Johnson, Staffan Larsson, and Aarne Ranta. The TALK Library: an Integration of GF with TrindiKit. Technical report, TALK deliverable 1.1, 2005.

Karen E. Lochbaum. A Collaborative Planning Model of Intentional Structure. *Computational Linguistics*, 24(4):525–572, 1998.

Markus Löckelt. Dialogue Management in the SmartKom System. In E. Buchberger, editor, *Proceedings of KONVENS 2004*, pages 125–132, Vienna, Austria, 2004.

Markus Löckelt. Action Planning for Virtual Human Performances. In *Proceedings of the 3rd International Conference on Virtual Storytelling*, pages 53–62, Strasbourg, France, 2005. Springer.

Markus Löckelt. Plan-Based Dialogue Management for Multiple Cooperating Applications. In Wolfgang Wahlster, editor, *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 301–316. Springer, Berlin, Germany, 2006.

Markus Löckelt and Norbert Pfleger. Multi-Party Interaction with Self-Contained Virtual Characters. In *Proceedings of the 9th Workshop on the Semantics and Pragmatics of Dialogue (DIALOR)*, pages 139–142, Nancy, France, 2005.

Markus Löckelt and Norbert Pfleger. Augmenting Virtual Characters for More Natural Interaction. In *Proceedings of the 3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2006)*, pages 231–240, Darmstadt, Germany, 2006.

Markus Löckelt, Tilman Becker, Norbert Pfleger, and Jan Alexandersson. Making Sense of Partial. In Johan Bos, Mary Ellen Foster, and Colin Matheson, editors, *Proceedings of the 6th Workshop on the Semantics and Pragmatics of Dialogue*, pages 101–107, Edinburgh, UK, 2002.

Markus Löckelt, Elsa Pecourt, and Norbert Pfleger. Balancing Narrative Control and Autonomy for Virtual Characters in a Game Scenario. In Mark T. Maybury, Oliviero Stock, and Wolfgang Wahlster, editors, *Proceedings of the First International Conference on Intelligent Technologies for Interactive Entertainment (INTETAIN) 2005*, pages 248–252, Madonna di Campiglio, Italy, 2005. Springer.

Markus Löckelt, Norbert Pfleger, and Norbert Reithinger. Multiparty Conversation for Mixed Reality. *International Journal of Virtual Reality*, 6(4):31–42, December 2007.

Brian Magerko, John E. Laird, Mazin Assanie, Alex Kerfoot, and Devvan Stokes. AI Characters and Directors for Interactive Computer Games. In *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*, pages 877–883, San Jose, CA, USA, 2004. AAAI Press.

William C. Mann. Dialogue Macrogame Theory. In *Proceedings of the 6th Workshop on Semantics and Pragmatics of Dialogue (EDILOG)*, pages 109–116, Edinburgh, UK, 2002.

William C. Mann. Dialogue Games: Conventions of Human Interaction. *Argumentation*, 2: 512–532, 1988.

Viviana Mascardi, Valentina Cordi, and Paolo Rosso. A Comparison of Upper Ontologies. Technical Report DISI-TR-06-21, University of Genova, Department for Informatics and Information Science, Genova, Italy, 2007.

Michael Mateas. *Interactive Drama, Art, and Artificial Intelligence*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2002.

Michael Mateas and Andrew Stern. Towards Integrating Plot and Character for Interactive Drama. In *Working Notes of the Social Intelligent Agents: The Human in the Loop Symposium*, AAAI Fall Symposium Series, pages 113–118, Menlo Park, CA, USA, 2000. AAAI Press.

Michael Mateas and Andrew Stern. Façade: An Experiment in Building a Fully-Realized Interactive Drama. In *Game Developers Conference, Game Design track (Online Proceedings)*, San Jose, CA, USA, March 2003.
Available at *http://citeseer.ist.psu.edu/mateas03facade.html* (last access 07.03.2008).

Mark T. Maybury and Wolfgang Wahlster, editors. *Readings in Intelligent User Interfaces*, chapter Introduction. Morgan Kaufmann Publishers, 1998.

Peter McBurney and Simon Parsons. Dialogue Games in Multi-Agent Systems. *Informal Logic, Special Issue on Applications of Argumentation in Computer Science*, 22(3):257–274, 2002a.

Peter McBurney and Simon Parsons. Games That Agents Play: A Formal Framework for Dialogues Between Autonomous Agents. *Journal of Logic, Language and Information*, 11 (3):315–334, 2002b.

Drew McDermott, Mark Burstein, and Douglas Smith. Overcoming Ontology Mismatches in Transactions with Self-describing Agents. In *The Emerging Semantic Web: Selected Papers from the First Semantic Web Working Symposium*, pages 228–244. 2002.

Stanley Milgram. Behavioral Study of Obedience. *Journal of Abnormal and Social Psychology*, 67:371–378, 1963.

Robert B. Miller. Response Time in Man-Computer Conversational Transactions. In *Proceedings of the AFIPS Fall Joint Computer Conference*, volume 33, pages 267–277, 1968.

David Milward and Martin Beveridge. Ontology-based Dialogue Systems. In *Proceedings of the 3rd Workshop on Knowledge and Reasoning in Practical Dialogue Systems (IJCAI'03)*, pages 9–18, Acapulco, Mexico, 2003.

Masahiro Mori. Bukimi No Tani (The Uncanny Valley). *Energy*, 7:33–35, 1970.

Daniele Nardi and Ronald J. Brachman. An Introduction to Description Logics. In F. Baader, D. Calvanese, D.L. McGuiness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook*, pages 5–44. Cambridge University Press, Cambridge, UK, 2002.

Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. Wiliam Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20: 379–404, 2003.

Dana S. Nau, Stephen J. J. Smith, and Kutluhan Erol. Control Strategies in HTN Planning: Theory versus Practice. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/10th conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 1127–1133, Menlo Park, CA, USA, 1998.

Alassane Ndiaye and Anthony Jameson. Predictive Role Taking in Dialog: Global Anticipation Feedback Based on Transmutability. In Sandra Carberry and Ingrid Zukerman, editors, *Proceedings of the 5th International Conference on User Modeling (UM-96)*, pages 137–144, Kailua-Kona, HI, USA, 1996.

Alassane Ndiaye, Patrick Gebhard, Michael Kipp, Martin Klesen, Michael Schneider, and Wolfgang Wahlster. Ambient Intelligence in Edutainment: Tangible Interaction with Life-Like Exhibit Guides. In *Proceedings of the Conference on Intelligent Technologies for Interactive Entertainment (INTETAIN'05)*, pages 104–113, Madonna di Campiglio, Italy, 2005.

Minh Hoai Nguyen and Wayne R. Wobcke. Towards a General Implementation of SharedPlans. In *Proceedings of the 3rd Australian Undergraduate Students' Computing Conference*, pages 42–48, Canberra, Australia, 2005.

Georg Niklfeld, Robert Finan, and Michael Pucher. Architecture for Adaptive Multimodal Dialog Systems based on VoiceXML. In *EUROSPEECH-2001*, pages 2341–2344, 2001.

Mina Nikolova. Konzeption und Realisierung von Wissensbasis und narrativer Kontrolle für ein Storytelling-System mit multiplen Agenten. Student project report, DFKI GmbH, Saarbrücken, Germany, 2006.

Daniel Oberle, Anupriya Ankolekar, Pascal Hitzler, Philipp Cimiano, Michael Sintek, Malte Kiesel, Babak Mougouie, Stephan Baumann, Shankar Vembu, Massimo Romanelli, Paul Buitelaar, Ralf Engel, Daniel Sonntag, Norbert Reithinger, Berenike Loos, Hans-Peter Zorn, Vanessa Micelli, Robert Porzel, Christian Schmidt, Moritz Weiten, Felix Burkhardt, and Jianshen Zhou. DOLCE ergo SUMO: On Foundational and Domain Models in the SmartWeb Integrated Ontology (SWIntO). *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(3):156–174, 2007.

Sharon Oviatt. *Handbook of Human-Computer Interaction*, chapter Multimodal Interfaces. Lawrence Erlbaum, New Jersey, NJ, USA, 2002.

Sharon L. Oviatt. Ten Myths of Multimodal Interaction. *Communications of the ACM*, 42(11): 74–81, 1999.

Sharon L. Oviatt. Multimodal System Processing in Mobile Environments. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software Technology (UIST'2000)*, pages 21–30, New York, NY, USA, 2000. ACM Press.

Sharon L. Oviatt and Philip R. Cohen. Multimodal Interfaces That Process What Comes Naturally. *Communications of the ACM*, 43(3):45–53, 2000.

Ana Paiva. The Role of Tangibles in Virtual Storytelling. In Gérard Subsol, editor, *Proceedings of the 3rd International Conference on Virtual Storytelling (ICVS'05)*, Lecture Notes in Computer Science 3805, pages 225–228, Strasbourg, France, 2005. Springer.

Ana Paiva, Joao Dias, Daniel Sobral, Ruth Aylett, Polly Sobreperez, Sarah Woods, Carsten Zoll, and Lynne Hall. Caring for Agents and Agents that Care: Building Empathic Relations with Synthetic Agents. In *AAMAS '04: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 194–201, Washington, DC, USA, 2004. IEEE Computer Society.

Hans Madsen Pedersen. Speech Acts and Agents. A Semantic Analysis. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 2002.

Christopher Peters, Simon Dobbyn, Brian MacNamee, and Carol O'Sullivan. Smart Objects for Attentive Agents. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media*, pages 1–8, Plzen, Czech Republic, 2003.

Norbert Pfleger. *FADE – an Integrated Approach to Multimodal Fusion and Discourse Processing*. PhD thesis, University of the Saarland, Saarbrücken, Germany, 2007.

Norbert Pfleger and Markus Löckelt. A Comprehensive Context Model for Multi-Party Interactions with Virtual Characters. In Jonathan Gratch, Michael Young, Ruth Aylett, Daniel Ballin, and Patrick Olivier, editors, *Proceedings of the 6th International Conference on Intelligent Virtual Agents (IVA)*, volume 4133 of *Lecture Notes in Computer Science*, pages 157–168, Marina del Rey, CA, USA, 2006. Springer.

Norbert Pfleger and Markus Löckelt. Synchronizing Dialogue Contributions of Human Users and Virtual Characters in a Virtual Reality Environment. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Interspeech/Eurospeech)*, pages 2773–2776, Lisbon, Portugal, 2005.

Norbert Pfleger, Jan Alexandersson, and Tilman Becker. Scoring Functions for Overlay and their Application in Discourse Processing. In *Proceedings of KONVENS'02 (online)*, Saarbrücken, Germany, 2002.
Available at *http://konvens2002.dfki.de/cd* (last accessed 07.12.2007).

Norbert Pfleger, Jan Alexandersson, and Tilman Becker. A Robust and Generic Discourse Model for Multimodal Dialogue. In *Workshop Notes of the IJCAI-03 Workshop on "Knowledge and Reasoning in Practical Dialogue Systems"*, pages 64–70, Acapulco, Mexico, 2003.

Luis Villaseñor Pineda, Antonio Massé Márquez, and Luis Alberto Pineda Cortés. A Multimodal Dialogue Contribution Coding Scheme. In *Proceedings of the Workshop on Meta-Descriptions and Annotation Schemes for Multimodal Language Resources, 2nd International Conference on Language Resources and Evaluation (LREC'2000)*, pages 52–56, Athens, Greece, 2000.

Manfred Pinkal. On Semantic Underspecification. In H. Bunt and Reinhard Muskens, editors, *Computing Meaning*, pages 33–56, Tilburg University, The Netherlands, 1999. Kluwer.

Massimo Poesio and David R. Traum. Towards an axiomatization of dialogue acts. In J. Hulstijn and A. Nijholt, editors, *Proceedings of TWENDIAL, the Twente Workshop on the Formal Semantics and Pragmatics of Dialogues*, pages 207–222, Enschede, the Netherlands, 1998.

Peter Poller and Valentin Tschernomas. Multimodal Fission and Media Design. In Wolfgang Wahlster, editor, *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 379–400. Springer, Berlin, Germany, 2006.

Robert Porzel, Norbert Pfleger, Stefan Merten, Markus Löckelt, Iryna Gurevych, Ralf Engel, and Jan Alexandersson. More on Less: Further Applications of Ontologies in Multi-Modal Dialogue Systems. In *3rd Workshop on Knowledge and Reasoning in Practical Dialogue Systems (IJCAI'03)*, pages 1–8, Acapulco, Mexico, 2003.

Vladimir A. Propp. *Morphology of the Folktale*. University of Texas Press, Austin, TX, USA, 1968.

Stephen G. Pulman. Conversational Games, Belief Revision and Bayesian Networks. In *CLIN VII: 7th Computational Linguistics in the Netherlands meeting*, pages 1–25, IPO, Technische Universiteit Eindhoven, the Netherlands, 1996.

Anand S. Rao and Michael P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann Publishers Inc.: San Mateo, CA, USA, 1991.

Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge, UK, 1999.

Norbert Reithinger and Daniel Sonntag. An Integration Framework for a Mobile Multimodal Dialogue System Accessing the Semantic Web. In *Proceedings of 9th European Conference on Speech Communication and Technology (Interspeech 2005)*, pages 841–844, Lisboa, Portugal, 2005.

Norbert Reithinger, Jan Alexandersson, Tilman Becker, Anselm Blocher, Ralf Engel, Markus Löckelt, Jochen Müller, Norbert Pfleger, Peter Poller, Michael Streit, and Valentin Tschernomas. SmartKom – Adaptive and Flexible Multimodal Access to Multiple Applications. In Sharon L. Oviatt, Trevor Darrell, Mark T. Maybury, and Wolfgang Wahlster, editors, *Proceedings of the 5th International Conference on Multimodal Interfaces (ICMI)*, pages 101–108, Vancouver, Canada, 2003.

Norbert Reithinger, Simon Bergweiler, Ralf Engel, Gerd Herzog, Norbert Pfleger, Massimo Romanelli, and Daniel Sonntag. A Look Under the Hood – Design and Development of the

First SmartWeb System Demonstrator. In *Proceedings of the 7th International Conference on Multimodal Interfaces (ICMI)*, pages 159–166, Trento, Italy, 2005.

Norbert Reithinger, Patrick Gebhard, Markus Löckelt, Alassane Ndiaye, Norbert Pfleger, and Martin Klesen. VirtualHuman – Dialogic and Affective Interaction with Virtual Characters. In *Proceedings of the 8th International Conference on Multimodal Interfaces (ICMI 2006)*, pages 51–58, Banff, Canada, 2006.

Charles Rich and Candace L. Sidner. COLLAGEN: A Collaboration Manager for Software Interface Agents. *User Modeling and User-Adapted Interaction*, 8(3-4):315–350, 1998.

Mark O. Riedl. Towards Integrating AI Story Controllers and Game Engines: Reconciling World State Representations. In *Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 84–89, Edinburgh, UK, 2005.

Mark O. Riedl and Andrew Stern. Believable Agents and Intelligent Story Adaptation for Interactive Storytelling. In *Proceedings of the 3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE'06)*, pages 1–12, Darmstadt, Germany, 2006a.

Mark O. Riedl and Andrew Stern. Failing Believably: Toward Drama Management with Autonomous Actors in Interactive Narratives. In *Proceedings of the 3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE06)*, pages 195–206, Darmstadt, Germany, 2006b.

Mark O. Riedl, Cesare J. Saretto, and R. Michael Young. Managing Interaction between Users and Agents in a Multiagent Storytelling Environment. In *Proceedings of the 2nd International Conference on Autonomous Agents and Multi-Agent Systems*, pages 741–748, Melbourne, Australia, 2003.

Thomas Rist, Stephan Baldes, Patrick Gebhard, Michael Kipp, Martin Klesen, Peter Rist, and Markus Schmitt. CrossTalk: An Interactive Installation with Animated Presentation Agents. In *Proceedings of the 2nd Conference on Computational Semiotics for Games and New Media (COSIGN 2002)*, pages 61–67, Augsburg, Germany, 2002.

Massimo Romanelli. Ontology-based Representation and Processing of Plurals for Human-Machine Dialogue Systems with Unification-based Operations. Master's thesis, University of the Saarland, Saarbrücken, Germany, 2005.

Martin Rumpler. *Statusbasierte Verhaltenssteuerung von virtuellen Charakteren*. PhD thesis, University of the Saarland, Saarbrücken, Germany, 2007.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, USA, 1995.

Harvey Sacks, Emanuel A. Schegloff, and Gail Jefferson. A Simplest Systematics for the Organization of Turn-Taking for Conversation. *Language*, 50:696–735, 1974.

Emanuel A. Schegloff and Harvey Sacks. Opening up Closings. *Semiotica*, 8:289–327, 1973.

Florian Schiel. Evaluation of Multimodal Dialogue Systems. In Wolfgang Wahlster, editor, *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 617–643. Springer Verlag, Berlin, Germany, 2006.

Marc Schröder and Jürgen Trouvain. The German Text-to-Speech Synthesis System MARY: A Tool for Research, Development and Teaching. *International Journal of Speech Technology*, 6:365–377, 2003.

John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, UK, 1969.

John R. Searle. A Taxonomy of Illocutionary Acts. In K. Gunderson, editor, *Language, Mind, and Knowledge*. University of Minnesota Press, Minneapolis, MN, USA, 1975.

John R. Searle and Daniel Vanderveken. *Foundations of Illocutionary Logic*. Cambridge University Press, Cambridge, UK, 1985.

Candace L. Sidner. An Artificial Discourse Language for Collaborative Negotiation. In *AAAI '94: Proceedings of the 12th National Conference on Artificial Intelligence (vol. 1)*, pages 814–819, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

Teresa Sikorski and James F. Allen. A Task-Based Evaluation of the TRAINS-95 Dialogue System. In *ECAI'96 Workshop on Dialogue Processing in Spoken Language Systems, revised papers*, number 1236 in Springer Lecture Notes in Computer Science, pages 207–220, Budapest, Hungary, 1997.

Mel Slater, Angus Altley, Adam Davison, David Swapp, Christoph Guger, Chris Barker, Nancy Pistrang, and Maria V. Sanchez-Vives. A Virtual Reprise of the Stanley Milgram Obedience Experiments. *PLoS ONE*, 1(1):e39, 2006.

Daniel Sonntag and Massimo Romanelli. A Multimodal Result Ontology for Integrated Semantic Web Dialogue Applications. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 511–516, 2006.

Daniel Sonntag, Ralf Engel, Gerd Herzog, Alexander Pfalzgraf, Norbert Pfleger, Massimo Romanelli, and Norbert Reithinger. SmartWeb Handheld – Multimodal Interaction with Ontological Knowledge Bases and Semantic Web Services. In *LNAI Special Volume on Human Computing*, pages 272–295. Springer, 2007.

William R. Swartout, Jonathan Gratch, Randy Hill, Ed Hovy, R. Lindheim, Stacy Marsella, Jeff Rickel, and David R. Traum. Simulation meets Hollywood: Integrating Graphics, Sound, Story and Character for Immersive Simulation. In Oliviero Stock and Massimo Zancanaro, editors, *Multimodal Intelligent Information Presentation*, pages 279–304. Springer, 2005.

William R. Swartout, Jonathan Gratch, Randall W. Hill, Eduard Hovy, Stacy Marsella, Jeff Rickel, and David R. Traum. Toward Virtual Humans. *AI Magazine*, 27(2):96–108, 2006.

Jürgen te Vrugt. *A Dynamic Multi-Application Dialog Engine for Task-Oriented Voice User Interfaces*. PhD thesis, University of the Saarland, Saarbrücken, Germany, 2006.

David R. Traum. 20 Questions on Dialogue Act Taxonomies. *Journal of Semantics*, 17(1): 7–30, 2000.

David R. Traum. Issues in Multi-Party Dialogues. In Frank Dignum, editor, *Advances in Agent Communication*, pages 201–211. Springer, Berlin/Heidelberg, Germany, 2004.

David R. Traum. Mental State in the TRAINS-92 Dialogue Manager. In *Working Notes of the AAAI Spring Symposium on Reasoning about Mental States: Formal Theories and Applications*, pages 143–149, 1993.

David R. Traum and James F. Allen. Discourse Obligations in Dialogue Processing. In *Proceedings of ACL-94*, pages 1–8, Las Cruces, NM, USA, 1994.

David R. Traum and Elizabeth A. Hinkelman. Conversation Acts in Task-Oriented Spoken Dialogue. *Computational Intelligence*, 8(3):575–599, 1992. Special Issue on Non-literal Language.

David R. Traum and Staffan Larsson. The Information State Approach to Dialogue Management. In Smith and Kuppevelt, editors, *Current and New Directions in Discourse & Dialogue*, pages 325–353. Kluwer, Dordrecht, the Netherlands, 2003.

David R. Traum and Jeff Rickel. Embodied Agents for Multi-Party Dialogue in Immersive Virtual Worlds. In C. Pelachaud and I. Poggi, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 766–773, Bologna, Italy, 2002.

Edward Tse, Saul Greenberg, Chia Shen, and Clifton Forlines. Multimodal Multiplayer Tabletop Gaming. In *Proceedings of the 3rd International Workshop on Pervasive Gaming Applications (PerGames)*, pages 141–150, Dublin, Ireland, 2006.

W3C-VoiceXML. Voice Extensible Markup Language (VoiceXML) 2.1, W3C Working Draft 15, 2006. (*http://www.w3.org/TR/voicexml21/*, last accessed Dec 20, 2006).

Wolfgang Wahlster. SmartKom: Fusion and Fission of Speech, Gestures, and Facial Expressions. In *Proceedings of the 1st International Workshop on Man-Machine Symbiotic Systems*, pages 213–225, Kyoto, Japan, 2002.

Wolfgang Wahlster. SmartKom: Symmetric Multimodality in an Adaptive and Reusable Dialogue Shell. In R. Krahl and D. Günther, editors, *Proceedings of the Human Computer Interaction Status Conference 2003*, pages 47–62, DLR: Berlin, Germany, 2003a.

Wolfgang Wahlster. Towards Symmetric Multimodality: Fusion and Fission of Speech, Gesture, and Facial Expression. In Andreas Günter, Rudolf Kruse, and Bernd Neumann, editors, *KI 2003: Advances in Artificial Intelligence. Proceedings of the 26th German Conference on Artificial Intelligence*, Springer LNAI, pages 1–18, Hamburg, Germany, 2003b.

Wolfgang Wahlster. SmartWeb: Towards Semantic Web Services for Ambient Intelligence (Presentation). In *Proceedings of the International Symposium on Life-World Semantics and Digital City Design*, Kyoto, Japan, 2004.

Wolfgang Wahlster. Overview of the Current State of the Lead Innovation VirtualHuman. In Bernd Reuse, Wolfgang Wahlster, and José L. Encarnação, editors, *VirtualHuman: Slides of the 3rd Project Meeting and VirtualHuman Network of Excellence Meeting held in conjunction with INTETAIN 2005*, Madonna di Campiglio, Italy, 2005.

Wolfgang Wahlster, editor. *Smartkom. Foundations of Multimodal Dialogue Systems*. Cognitive Technologies Series. Springer, Berlin, Germany, 2006.

Wolfgang Wahlster and Alfred Kobsa. *User Models in Dialog Systems*, chapter User Models in Dialog Systems, pages 4–34. Springer, Berlin, Germany, 1989.

Wolfgang Wahlster, Norbert Reithinger, and Anselm Blocher. SmartKom: Multimodal Communication with a Life-Like Character. In *Proceedings of Eurospeech 2001, 7th European Conference on Speech Communication and Technology*, volume 3, pages 1547–1550, Aalborg, Denmark, 2001.

Marilyn Walker, Candace A. Kamm, and Diane J. Litman. Towards Developing General Models of Usability with PARADISE. *Natural Language Engineering*, 6(3-4):363–377, 2000.

Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. PARADISE: A Framework for Evaluating Spoken Dialogue Agents. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 271–280, Somerset, New Jersey, USA, 1997.

Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter Belief-Desire-Intention Architectures, pages 54–61. MIT Press, 1999.

Joseph Weizenbaum. ELIZA – A Computer Program For the Study of Natural Language Communication Between Man And Machine. *Communications of the ACM*, 9(1):36–45, 1966.

Ludwig Wittgenstein. *Philosophische Untersuchungen*. Suhrkamp Verlag, Frankfurt am Main, Germany, 1953.

R. Michael Young. An Overview of the Mimesis Architecture: Integrating Intelligent Narrative Control into an Existing Gaming Environment. In *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, pages 77–81, Menlo Park, CA, USA, 2001. AAAI Press.