UNIVERSITÄT
DES
SAARLANDES

SiAM-dp: An open development platform for massively multimodal dialogue systems in cyber-physical environments

Robert Neßelrath

Thesis for obtaining the title of
Doctor of Engineering Science
of the Faculty of Natural Science and Technology I
of Saarland University

Saarbrücken, December 17, 2015

Tag des Kolloquiums:
27. Januar 2016

Dekan:
Prof. Dr. Markus Bläser

Berichterstatter:
Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster
Prof. Dr. Wolfgang Maaß

Vorsitzender:
Prof. Dr. Jörg Hoffmann

Akademischer Mitarbeiter:
Dr. Michael Feld

Danksagung

Eine Reihe von Personen hat zur Entstehung dieser Arbeit sowohl durch fachliche als auch moralische Unterstützung beigetragen. An dieser Stelle möchte ich hierfür meinen herzlichen Dank aussprechen.

An erster Stelle gilt dieser Dank meinem Doktorvater Prof. Wolfgang Wahlster, der es mir überhaupt ermöglicht hat, in seiner Forschungsgruppe für intelligente Benutzerschnittstellen am Deutschen Forschungszentrum für künstliche Intelligenz (DFKI), im Rahmen meiner Tätigkeiten als Forscher, diese Doktorarbeit zu verfassen. Besonders die Betreuung in der Endphase war sehr intensiv und hat es mir ermöglicht, in einer konstruktiven und angenehmen Arbeitsatmosphäre der Arbeit den finalen Feinschliff zu geben. Bei Prof. Wolfgang Maaß möchte ich mich für die kurzfristige Bereitschaft zum Verfassen des Zweitgutachtens bedanken.

Ein besonderes Augenmerk gilt meinen Arbeitskollegen am DFKI, die zu einem spannenden und inspirierenden Arbeitsumfeld beigetragen haben. Da ich nach Abgabe meiner Arbeit das DFKI verlassen werde, möchte ich mich für 7,5 Jahre der angenehmen und freundlichen Zusammenarbeit bedanken. Dies betrifft insbesondere die vielen Kollegen, mit denen ich über die Jahre in den Projekten i2home, Theseus, Inwero und SemProM zusammenarbeiten durfte. Meinen Kollegen in den Projekten SiAM und MadMacs, mit denen ich gemeinsam mehrere der Demonstratoren zu dieser Arbeit gebaut habe, ein großes Dankeschön für die schöne Zeit: Michael Feld, Yannick Körber, Mohammad Mehdi Moniri, Monika Pepik und Tim Schwartz.

Ein ganz besonderer Dank gilt meinen 'Leidensgenossen', den ehemaligen und aktuellen Doktoranden, mit denen ich durch unterstützende Diskussionen, besonders in der langwierigen Phase zum Ende hin, die Motivation aufrechterhalten konnte. Namentlich möchte ich hier Matthieu Deru, Jochen Frey und Daniel Porta nennen.

Für die sprachliche Korrektur bedanke ich mich bei Sylvia Krüger und Kate Flynn Rau.

Von ganzem Herzen bedanke ich mich bei meiner Familie, Prisca und meinen Freunden, die mich während der Erstellung dieser Arbeit - von nah und fern - über all die Jahre nach Kräften unterstützt, gefördert und für die nötige Balance gesorgt haben.

Zusammenfassung

Cyber-physische Umgebungen (CPEs) erweitern natürliche Alltagsumgebungen wie Heim, Fabrik, Büro und Auto durch Verbindung der kybernetischen Welt der Computer und Kommunikation mit der realen, physischen Welt. Die möglichen Anwendungsgebiete hierbei sind weitreichend. Während unter dem Stichwort *Industrie 4.0* cyber-physische Umgebungen eine bedeutende Rolle für die nächste industrielle Revolution spielen werden, erhalten sie ebenfalls Einzug in Heim, Büro, Werkstatt und zahlreiche weitere Bereiche. In solch einer neuen Welt geraten klassische Interaktionskonzepte, in denen Benutzer ausschließlich mit einem einzigen Gerät, PC oder Smartphone interagieren, immer weiter in den Hintergrund und machen Platz für eine neue Ausprägung der Interaktion zwischen dem Menschen und der Umgebung selbst. Darüber hinaus sorgen neue Technologien und ein wachsendes Spektrum an einsetzbaren Modalitäten dafür, dass sich im Interaktionsdesign neue Möglichkeiten für eine natürlichere und intuitivere verbale und nonverbale Kommunikation auftun.

Die dynamische Natur von cyber-physischen Umgebungen und die Mobilität der Benutzer darin stellt Anwendungsentwickler vor die Herausforderung, Systeme zu entwickeln, die flexibel bezüglich der verbundenen und verwendeten Geräte und Modalitäten sind. Dies impliziert auch neue Möglichkeiten in der modalitätsübergreifenden Kommunikation, die über duale Interaktionskonzepte, wie sie heutzutage bereits üblich sind, hinausgehen.

Die vorliegende Arbeit befasst sich mit der Unterstützung von Anwendungsentwicklern mit Hilfe einer Plattform zur deklarativen und modellbasierten Entwicklung von multimodalen Dialogapplikationen mit einem Fokus auf verteilte Ein- und Ausgabegeräte in cyber-physischen Umgebungen. Die bearbeiteten Aufgaben können grundlegend in drei Teile gegliedert werden:

- Die Konzeption von Modellen und Strategien für die Spezifikation von Dialoganwendungen in einem deklarativen Entwicklungsansatz. Dies beinhaltet Modelle für das Definieren von Projektressourcen, Dialogverhalten, Spracherkennergrammatiken, graphischen Benutzerschnittstellen und Abbildungsregeln, die die gerätespezifische Darstellung von Ein- und Ausgabegeräten in eine gemeinsame Repräsentationssprache transformieren.
- Die Implementierung einer Laufzeitumgebung, die eine flexible und erweiterbare Architektur für die einfache Integration neuer Geräte und Komponenten bietet. Die Plattform realisiert Konzepte und Strategien der multimodalen Mensch-Maschine-Interaktion und ist die Basis vollwertiger multimodaler Dialoganwendungen für beliebige Domänen, Szenarien und Gerätekonfigurationen.
- Eine Softwareentwicklungsumgebung, die in die Eclipse Rich Client Plattform integriert ist und Entwicklern Assistenten und Editoren an die Hand gibt, die das Erstellen und Editieren von neuen multimodalen Dialoganwendungen unterstützen.

Abstract

Cyber-physical Environments (CPEs) enhance natural environments of daily life such as homes, factories, offices, and cars by connecting the cybernetic world of computers and communication with the real physical world. While under the keyword of *Industrie 4.0*, CPEs will take a relevant role in the next industrial revolution, and they will also appear in homes, offices, workshops, and numerous other areas.

In this new world, classical interaction concepts where users exclusively interact with a single stationary device, PC or smartphone become less dominant and make room for new occurrences of interaction between humans and the environment itself. Furthermore, new technologies and a rising spectrum of applicable modalities broaden the possibilities for interaction designers to include more natural and intuitive non-verbal and verbal communication.

The dynamic characteristic of a CPE and the mobility of users confronts developers with the challenge of developing systems that are flexible concerning the connected and used devices and modalities. This implies new opportunities for cross-modal interaction that go beyond dual modalities interaction as is well known nowadays.

This thesis addresses the support of application developers with a platform for the declarative and model based development of multimodal dialogue applications, with a focus on distributed input and output devices in CPEs. The main contributions can be divided into three parts:

- Design of models and strategies for the specification of dialogue applications in a declarative development approach. This includes models for the definition of project resources, dialogue behaviour, speech recognition grammars, and graphical user interfaces and mapping rules, which convert the device specific representation of input and output description to a common representation language.
- The implementation of a runtime platform that provides a flexible and extendable architecture for the easy integration of new devices and components. The platform realises concepts and strategies of multimodal human-computer interaction and is the basis for full-fledged multimodal dialogue applications for arbitrary device setups, domains, and scenarios.
- A software development toolkit that is integrated in the Eclipse rich client platform and provides wizards and editors for creating and editing new multimodal dialogue applications.

Contents

List of Figures

List of Tables

List of Acronyms

ABNF	Augmented Backus-Naur-Form
BCI	Brain-Computer Interface
CFG	context-free grammar
CPE	Cyber-physical Environment
CPS	Cyber-physical System
CSS	Cascading Style Sheet
EMF	Eclipse Modeling Framework
EMMA	Extensible MultiModal Annotation markup language
eTFS	extended Typed Feature Structure
FADE	Fusion and Discourse Engine
FSA	Finite state automaton
GRXML	Speech Recognition Grammar Specification
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HEI	Human-Environment Interaction
LTM	Long-Term Memory
MUI	Multimodal User Interface
NLP	Natural Language Processing

ODP	Ontology-based Dialog Platform
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SCXML	State Chart XML
SiAM-dp	Situation Adaptive Multimodal Dialogue Platform
SRGS	Speech Recognition Grammar Specification
SSML	Speech Synthesis Markup Language
SWEMMA	SmartWeb EMMA
TFS	Typed Feature Structure
URI	Uniform Resource Identifier
WM	Working Memory

1 Introduction

1.1 Motivation

Intelligent environments enhance natural environments of daily life such as homes, factories, offices, and cars by bringing computation into the physical world. Recent advances in technologies like microprocessors, networks, middleware, sensors, and mobile devices increasingly push the evolution of ubiquitous computing where technology is omnipresent and wirelessly connected processing units can be found in arbitrary everyday objects. Thus, technology is increasingly embedded into the environment, obfuscating the underlying IT technology from users. Especially in the context of industry and production, the term CPE has been established, which connects the cybernetic world of computer and communication with the real world where intelligent objects and agents can communicate with each other. This is a starting position for a wide range of application areas like intelligent factories, intelligent buildings, intelligent retail shops, and intelligent mobility solutions, which will have a high impact on society and industry in the future.

The emergence of such CPEs requires a complete rethinking of the interaction between the environment and the users that become part of the environment, move in it, and interact with it. The classical interaction concepts where a user exclusively interacts with one stationary device, PC or smartphone, become less dominant and make room for concepts where the user interacts with the environment (Human-Environment Interaction (HEI)). Novel interface technologies like speech interaction, gesture recognition, distributed displays, eye-tracking, and wearable devices reinforce the trend to carry the interaction away from stationary places where keyboards, mice, buttons, and switches are the usual input devices, into the room.

In a heavily instrumented environment with a high number of various sensors and actuators it becomes possible to address all human senses in a multimodal manner. This provides the opportunity of multiadaptive systems, which are precisely customised and personalised to the requirements of the users and contextual conditions. Since new technologies and the availability of a great spectrum of communication through numerous modalities enable the use of a large number of verbal and non-verbal human communication characteristics, a great opportunity is offered to design the interaction between human and the environment more naturally and thus more intuitively.

With new opportunities researchers also face new challenges regarding the applied interaction technologies. Due to the dynamic characteristic of a CPE and the mobility of the users, they need to develop systems that are flexible concerning the connected and used devices and modalities. Furthermore, multiple users may simultaneously interact with the environment either jointly in a dialogue with a common intention or simultaneously in parallel and independent sessions. Besides that, even physical acts may be part of interactions and can play a key role, e.g., for collaborative task solving or capturing the attention of the user. The new chances of multimodal interaction go beyond modality combinations with up to two or three diverse modalities in previous research, but at the same time raise the question of how information and meaning of the individual contributions should be practically combined. Further research is heading into the direction in which interaction metaphors should be applied to present the CPE to the user. This could be a pervasive intelligence but alternatively distributed, even anthropomorphic interfaces. All of them could take on different roles in dialogue interaction; they can, depending on the use-case, act as a butler, supervisor or supporter.

Many former research projects addressed relevant topics that are considered to be fundamental preconditions for the above introduced aims. This includes multimodal integration, multimodal fusion, multimodal fission, dialogue management, discourse resolution, ontology based knowledge representation, and situation adaptive systems. Other important works deal with the representation of interaction and propose language models for the description of interaction on the levels of realisation and the communicative meaning behind. Furthermore, in literature one can find a significant number of prototypes and demonstration systems that integrate many kinds of modalities and devices in a wide range of scenarios and domains.

However, most of the projects focus on concrete research questions and provide prototypical, often 'hard-wired' software solutions for individual challenges. More scenario oriented projects highly concentrate on usability without attaching importance to the reusability of technology in other domains or extendability with new interfaces and devices. Some existing toolkits for the development and deployment of multimodal interfaces present promising design approaches, but only integrate solutions with two modalities like speech together with graphical user interfaces or pointing gestures. Other extensive projects deploy well elaborated platforms for multimodal dialogue systems. Unfortunately they are not sustainable due to the lack of maintenance, open accessibility or they are built upon out-of-date software engineering technologies. Furthermore, often toolkits that support the work of application developers are missing.

A base requirement for the research regarding interaction in CPE is a platform that is able to handle multimodal interaction to a high degree. This includes, on the one hand, a technical architecture for the flexible management of connected devices that is able to efficiently and robustly distribute input and output events. On the other hand, a modelling language is required that hides the heterogeneity of devices, technologies, protocols and information behind a sophisticated and extendable approach that additionally allows one to semantically represent knowledge entities and communicative intentions. Standard modalities like speech recognition, speech synthesis and graphical user interfaces should be accessible out-of-the-box by declarative development concepts. The rapid integration of new devices should be realisable by supporting design methodologies and functionalities. Furthermore, concepts and technological solutions for multimodal dialogue system aspects like fusion, fission, context management, and discourse resolution should be available in a comprehensive and easy to apply way. Research on interaction metaphors and their evaluation requires a concept for the rapid prototyping of dialogue strategies. This process should be aided by a declarative modelling language that is comprehensible even for non-expert dialogue engineers.

The SiAM-dp that has been developed for this thesis is intended to bridge this gap. It introduces a declarative model-based development approach that covers the above mentioned features on a conceptual as well as a functional level. Complementary to the runtime environment, the platform contains a toolkit integrated in the Eclipse development environment that supports the development, debugging and deployment of multimodal dialogue applications. In the following, some main features of SiAM-dp are listed.

Massively Multimodal Interaction

The most current systems in research mainly address scenarios that support multimodality as a combination of two different modalities. A dialogue management system for a CPE must be able to deal with massively multimodal interactions trying to concurrently address all human senses in heavily instrumented environments. On the one hand, this includes the *free choice of modality*, which means that any interaction should be, if possible, realisable by every modality available based on the preferences of the user. On the other hand, clearly more than two modalities are integrated into a multimodal system that are also used in combination. In Section 9.1, a dialogue application is presented that exemplarily combines gaze detection, speech input, and finger gestures together in a combined input. Massive modality also means that many homogeneous devices of the same modality are used together in one application. This can be several microphones that collect speech input commands from several users. Other examples are several tangible interfaces like buttons or levers in a car. In classical HMIs the 'hard-wired' connection between button and function is isomorphic. One button triggers exactly one function and one function can only be triggered by one specific button. In a massively multimodal system this isomorphic relation is decoupled and allows interface designers to trigger one function by an arbitrary modality. On the other way around the triggered function, when pressing a button, can be made dependent on the actual context, e.g., affected by a synchronous speech command or the situational context. SiAM-dp is capable of handling these scenarios in a highly flexible way.

Distributed Input and Output

SiAM-dp supports a wide range of ways to connect devices to the system. Thus, a multimodal dialogue application can distribute input and output over devices that are directly connected to the system on which the application is running, devices that are wirelessly connected via Ethernet or Bluetooth, over CANBus in a car, internet services, and even public displays or other cars that use Car2X technology (many examples are presented in Chapter 9).

Ontology Based

SiAM-dp is fully ontology based and uses a single domain adaptable knowledge representation throughout the complete dialogue system. This includes that interactions can be annotated with semantics, which provide information about communicative intentions and eventually transported semantic content. Thus, interactions can be described on two separate levels of abstraction, the above mentioned semantic intention and the actual surface realisation. The task of performing the step between these two levels is resolved by modality-specific interpreters on the input side and generators on the output side. The advantage of this approach is that multimodal late fusion, context resolution, dialogue planning, and presentation planning can be realised on the abstract semantic level, independent from the actually applied surface realisations.

Situation Adaptivity

SiAM-dp allows one to adjust dialogue behaviour and presentation dependent on the user, who is actually using the system, and the current context. This is supported by a central knowledge base that is accessible system-wide. An important factor for this feature is an architecture that decouples the decision about dialogue planning from the decisions about presentation planning. Furthermore, it is possible to build resource-aware systems that makes decisions based on the expected resource / cost impact (workload, distraction, time) under varying goals.

Software Development Kit

SiAM-dp contains a complete Software Development Kit (SDK) with a collection of tools and editors that support the developer in the creation of domain-specific dialogue applications. This includes elaborated development processes and strategies for the model-based development of applications. Furthermore, it supports the extension of the dialogue platform with new devices and thus allows one to easily integrate new modalities into the platform which can be reused in diverse dialogue applications.

1.2 Research Questions

The primary and overall aim of the work presented in this thesis is the development of a multiadaptive and massively multimodal dialogue platform by proposing a runtime platform and a development environment which support the rapid development of dialogue applications that are flexible, adaptable and extendable to scenarios in various domains. The concrete objectives of this work are manifold and include concepts and ideas from diverse research perspectives. First, the technical question for an architecture arises that robustly integrates and handles input and output from a heterogeneous set of devices and technologies. Second, information must be processed in a way that typical aspects from multimodal dialogues like mutual disambiguation, context representation, referring expressions and discourse resolution are supported. Furthermore, the fusion of information and the distribution of output play a relevant role. Finally, we aim at a comprehensive and easy-to-use declarative development approach that supports the user in creating specifications of dialogue behaviour, input interpretation, output generation and context description. The main outcome of this work is SiAM-dp, which has already been deployed in a first version and is successfully used by several research and industry projects in the intelligent user interfaces group of the German Research Center for Artificial Intelligence. Some of these system are presented in Chapter 9.

The following research questions are addressed in this dissertation:

1. ***Modelling Language:*** *Which requirements must be fulfilled by a meta-modelling language that is used in a declarative development approach for multimodal dialogue applications?*

   Different tasks demand different features from the meta modelling language. It must be expressive enough to represent semantic knowledge and interrelations between entities. Features for multimodal dialogue processing and context resolution need specific operations for reasoning purposes. From a technical point of view the model must be capable of being easily integrated into a comprehensive runtime environment. Finally, the meta-language should already support concepts for creating and editing models in an integrated development environment.

2. ***Massive Multimodality:*** *How can the massive modality of devices in a CPE be represented in a hierarchical device model and how can this hierarchy be transferred to a structured model for the representation of input and output acts in the communication between the dialogue system and devices?*

   CPEs can contribute a great number of accessible devices to multimodal dialogue applications. The selection of a suitable combination of devices that are actually involved in interaction is an essential question for situation adaptive dialogue systems. Therefore devices must be classified in terms of their type, the kind of information they provide or process, and other characteristic features like location and interaction range. This hierarchy should be universally transferred to the applied model for input and output representation in order to ensure a holistic approach for the representation of information.

3. ***Representation of Communicative Meaning:*** *How can interaction between dialogue systems and a highly heterogeneous set of devices in a CPE be represented independently of modality, and how can this support the multimodal integration?*

   Dialogue acts enrich a dialogue with new content but what is more important is their communicative meaning. Especially contributions in a natural language dialogue can be incomplete or ambiguous as long as context information from discourse and environment is not considered. Physical acts that occur simultaneously or contemporarily are part of this context. The joint integration of dialogue acts and physical acts requires that their content and containing (cross-modal) referring expressions are semantically represented in a common and consistent way. Especially in a massively multimodal system and, associated therewith, an enormous increase of available context information, the requirements to a language for the representation of input and output are increasing.

4. ***Declarative Dialogue Application Design:*** *Which dialogue application specification models support the rapid development of multimodal dialogue applications in Human-Computer Interactions (HCIs)?*

   The creation of multimodal dialogue applications requires a modelling approach that supports the development process and is flexible enough to easily adapt applications to new domains, device setups and situations. Several previous works have dealt with the creation of full-fledged languages for the description of multimodal interaction. Findings from these projects should be examined and transferred to a development platform for multimodal interaction in CPEs.

5. ***Dialogue System Architecture:*** *Which type of architecture is required for the realisation of distributed coordinated communication in an CPE?*

   The support of modalities and physical devices that are available at the time of implementation, as well as those that become available in the future, requires a modular and flexible platform architecture. Events that occur simultaneously or sequentially must be adequately time stamped and handled by components that are responsible for modality fusion. Since output can be distributed over several modalities the architecture must allow one to coordinate and time the presentation of information.

6. ***Tool Support:*** *How is an integrated development environment designed that simplifies and accelerates the creation of multimodal dialogue applications ?*

   A convincing and easy-to-use tool support is an important factor for the acceptance of a development framework. Recurring tasks should be simplified or taken over by wizards. Furthermore, the toolkit should organise application relevant resources in projects and deploy editors for the creation and manipulation of their content.

1.3 Chapter Outline

This dissertation is divided into three main parts, which are again structured into eight chapters. Figure 1.1 gives an overview of the structure of the work:

Foundations

Chapter 2 (Fundamental Concepts) presents general concepts and background information about the terminology used in this dissertation. This includes a basic understanding of human communication and multimodal HCI. This is followed by an introduction of the research topics *Dialogue Systems* and *Cyber-physical Environments*. Chapter 3 starts with related work in the field of multimodal interaction. It forms a bridge from the first pioneer works to modern multimodal dialogue applications in intelligent environments. Further sections in this chapter deal with multimodal dialogue development frameworks and modelling approaches for the representation of multimodal interaction and the semantic annotation of dialogue acts with communicative meaning.

Concepts

The second part presents the acquired concepts in this work within three chapters. Chapter 4 (The SiAM-dp modelling language) examines the requirements that must be fulfilled by the meta-modelling language, which is used in the dialogue platform. This includes a short excursion about semantic knowledge representation. Finally, the chapter describes the modelling language that has been chosen and explains which extensions were made in order to satisfy the identified requirements. Chapter 5 (Massive Multimodality in Cyber-Physical Systems) presents a classification approach for devices in massively multimodal CPEs, which is used for the creation of a uniform interface that helps one to consider the heterogeneity of devices during integration. Furthermore, the chapter describes in detail the model that has been developed for the representation of interaction based on the before elaborated device classification. Here a special focus is set on the support of a semantic level that additionally allows the modality-independent description of the intention behind an interaction. The chapter closes with the presentation of a rule-based approach that supports developers in easily performing the step between the syntactic and semantic representation of dialogue acts. Chapter 6 (Dialogue Application Specification Models) deals with the models that have been developed in this thesis for the declarative development of dialogue applications. This includes models for the specification of projects, dialogue behaviour, speech recognition grammars, and graphical user interfaces. Furthermore, a strategy is presented in order to specify model instances that are dynamically filled with content during runtime based on script expressions and the overlay algorithm.

Realisation

In the final part, the aforementioned concepts and models are realised and applied. Chapter 7 (SiAM Dialogue Platform) presents the architecture of the SiAM-dp runtime environment and gives detailed insight into event management and the individual components of the platform. Chapter 8 (Development Tools) describes the SiAM-dp workbench, an Eclipse based tool set for the development of multimodal dialogue applications. Finally, Chapter 9 (Applications) outlines prototypes and demonstrator applications that have been developed in several research projects on the basis of SiAM-dp. The chapter emphasises the great heterogeneous set of devices and modalities that already have been integrated into the platform.

Discussion

In Chapter 10 the results of this thesis are discussed with respect to the underlying research questions. Especially the scientific and practical contributions are highlighted. After an enumeration of all publications on international conferences and workshops that have been written in the context of this work, an outlook is given for future work that might extend the results of this thesis.

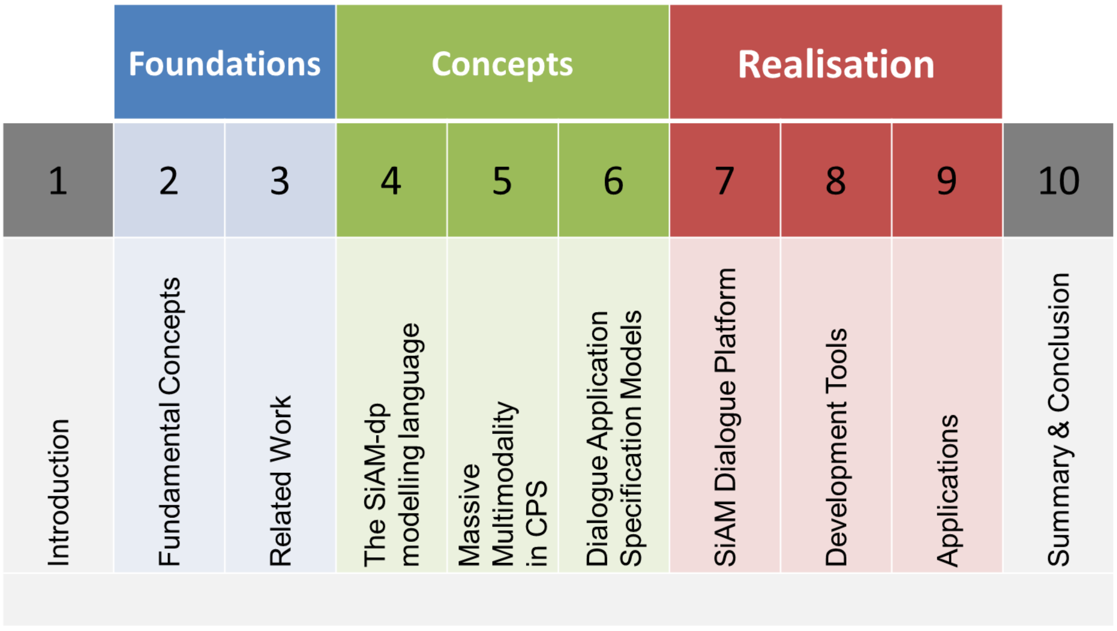


Figure 1.1 – The structural overview of the dissertation

2 Fundamental Concepts

This chapter introduces some fundamental concepts which provide the relevant foundation for the multimodal dialogue platform presented in this thesis. For a better understanding of human-computer interaction, we first take a closer look at human perception and communication in Section 2.1. A special focus is set on the interplay between verbal and nonverbal communication and the hereby involved modalities.

In the following we show how these concepts are adopted and adapted for the communication between human and computer. For the output, this affects how human senses are stimulated for the transfer of information; for the input how sensors equivalently to human senses can perceive human communication. Section 2.2 deals with multimodal communication and considers its advantages and the resulting challenges that arise.

Section 2.3 gives a short introduction to the idea of dialogue systems, several approaches of dialogue management, and finally the challenge of context resolution which increasingly appears with dialogue systems that support a multimodal architecture in Cyber-physical Environments (CPEs).

Since the platform has been developed for operation in CPEs, this concept is defined and explained in Section 2.4. This includes the typical structure, some relevant requirements, and the affect on human-computer interaction.

2.1 Human Perception and Communication

“The environmental stimulus is all of the things in our environment that we can potentially perceive” (Goldstein, 2009). Animals and humans perceive by the physical or chemical stimulation of sense organs that produce signals in the nervous system. Thus, the sense organs provide the data for estimating states of the environment and are a person’s interface to the world. In his treatise ‘De Anima’ Aristotle already described the five classic human senses: sight, hearing, touch, taste, and smell. This allegory of the five senses was very popular throughout the Middle Ages (see Figure 2.1). However, in modern science additional senses are accepted, e.g., thermoception(heat), nociception (physiological pain), equilibrioception (balance, acceleration), and proprioception (kinesthetic sense).

Originally, senses played an important role for collecting properties from the environment in order to gather food, recognise hazards, or find potential mates. Furthermore, senses helped to perceive the communicative behaviour of other individuals and thus made communication possible. Generally, multiple senses are employed while retrieving information from other individuals. Very common are sight and hearing, and nearly every kind of animal developed their own set of body gestures and sounds for intra-specific and inter-specific communication. In the course of evolution, other, more complex communication types have evolved. The apparently most manlike animal species in terms of communication are apes. Like humans, they apply facial expressions to show emotions. Gillespie-Lynch et al. (2013) describe a function and formal similarity of many gestures between a chimpanzee, a bonobo, and a human child, e.g., deictic gestures. With the modern human, a new form of communication, language, has evolved to be the primary mode of communication. While language was originally transferred by speech, with the appearance of human cultures, new ways for exchanging information were invented like written language and symbols.

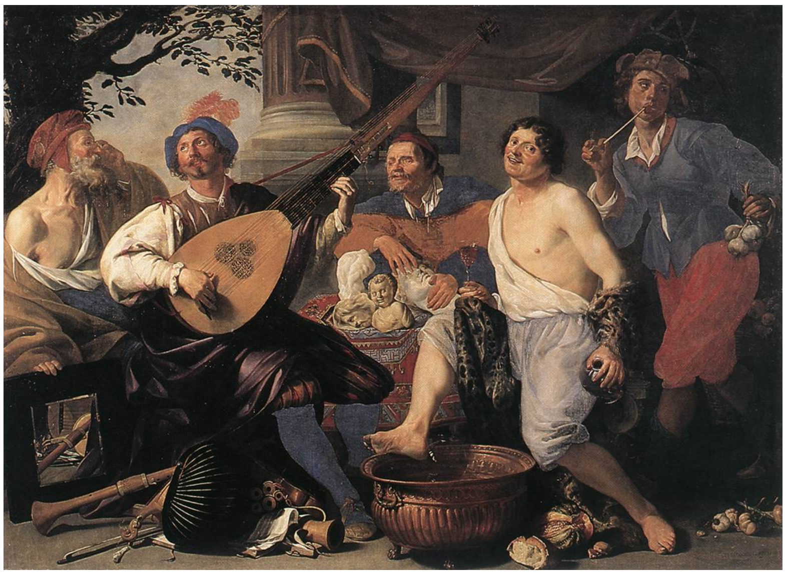


Figure 2.1 – The allegory of the Five Senses by Theodore Rombouts: From left to right the five senses are presented: sight, hearing, touch, taste, and smell.

2.1.1 Verbal and Nonverbal Communication

The word communication stems from the latin word *communicare*, which can be translated as *join with* or *share*. The National Joint Committee for the Communication Needs of Persons with Severe Disabilities (1992) defines communication as “any act by which one person gives to or receives from another person information about that person’s needs, desires, perceptions, knowledge, or affective states. Communication may be intentional or unintentional, may involve conventional or unconventional signals, may take linguistic or non-linguistic forms, and may occur through spoken or other modes”. In addition to the ‘one-to-one’ communication, as mentioned in the definition, communication can also be ‘one-to-many’ (giving a speech), ‘many-to-many’ (group conversations) or ‘one-to-many-undetermined-recipients’ (market-crier). The latter can also be called ‘now-to-future’ communication if the information is persistent (books, video talk) (Wasinger, 2006). With the increasing importance of computers and the aim to build intuitive and easy-to-use control interfaces, the concepts of human communication are adopted to the communication between computer and human. Thus, the previously mentioned definition can be extended to include machines as a potential communication participant (see Section 2.2).

Although verbal communication (spoken or written language) is the most expressive way to exchange information, a great amount of the content is conveyed by bodily activity, gesture, facial expression, posture and spacing, touch and smell, and of those aspects of spoken utterances that can be considered apart from the referential content of what is said (Ruesch and Kees, 1956; Kendon et al., 1981; Knapp and Hall, 2009). Wasinger (2006) provides a classification of communication types (Figure 2.2). On the top level, communication is categorized into *verbal* and *nonverbal* types. The verbal contribution consists of language in spoken or written form. Nonverbal communication is expressed by means others than words. The three major subcategories of nonverbal communication are *visual*, *auditory*, and *invisible* communication. Auditory information mostly appears with spoken language. By varying the acoustic properties of speech, e.g. speed, accentuation or volume, the speaker can embed additional messages into verbal content, like emotions or detailed information concerning the function of an utterance (irony, command, criticism). The invisible communication, e.g., contains *tactile* and *olfactory* communication. Finally, a huge amount of information is visually transferred, whereas *kinesic* communication makes up the largest part of it (body gestures, face expressions, eye behaviour). Furthermore, information like personal feelings, the social status of a contributor or his personal relationship to the dialogue partner can be expressed by the two other subclasses of visual communication, which are *proxemic* and *artefactual* communication.

In most cases, verbal and nonverbal communication operate together since a person constantly (intentionally and unintentionally) transmits messages with his body language. Knapp and Hall (2009) describe six ways of the interrelation between verbal and nonverbal communication that occur simultaneously:

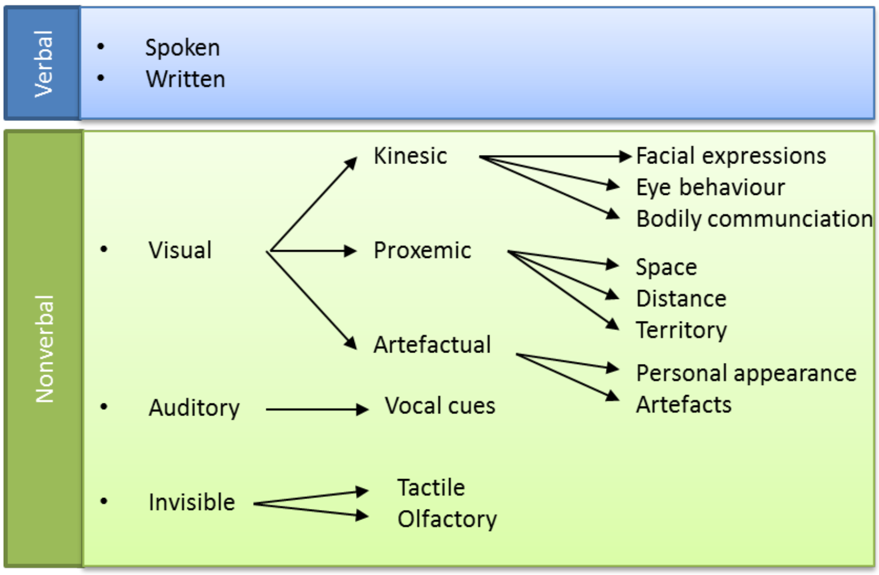


Figure 2.2 – Classification of verbal and nonverbal communication (Wasinger (2006))

1. *Substituting* - The nonverbal message is used in the place of a verbal message. This can be an iconic gesture like a raised thumb or a "stop" gesture. Facial expressions can provide information about the actual emotional state of a speaker without an additional verbal message.

2. *Repeating* - The content of the verbal channel is simply repeated and reinforces the meaning of the verbal messages, e.g., a nodding during an agreeing verbal utterance.

3. *Complementing* - A verbal utterance is enhanced with additional content, e.g., a pointing gesture that identifies an entity while the speaker asks for information about it.

4. *Contradicting* - Nonverbal and verbal messages can send contradicting information. The reasons for this can be identified in a mixed feeling about the content or in imperfect lying. While in these situations the nonverbal message is unconsciously done, the behaviour can also be planned, e.g., with a sarcastic tone of voice or a wink of the eye.

5. *Communication Flow Regulating* - Nonverbal behaviour is used to manage conversations, especially the turn-taking. The moderator of a discussion uses gestures in order to invite a contributor to talk. Also the tone of the voice indicates whether a speaker's contribution ends or if he wants to speak. If an answer is expected, the speaker directly makes eye contact with the other person.

6. *Accenting/Moderating* - Nonverbal behaviour can be used to emphasize or moderate parts of a verbal message.

Another interrelation, which is not mentioned here, is the *mutual disambiguation* that helps to correct unimodal recognition errors by using partial information provided by a second modality. Kelly et al. (1999) found out in four experiments that "speech and gesture may interactively contribute to the meaning of a communicative act" and thus help to correctly understand intended meanings. Oviatt (1999a) demonstrates that with mutual disambiguation, also multimodal systems can be designed to run in a more robust and stable manner. In a study she showed that within a multimodal architecture the total error rate could be reduced by 41% for spoken language processing, compared with standalone spoken language processing.

2.1.2 Human-Computer-Communication

The previous section discussed that humans intentionally and unintentionally address several of their senses to communicate with each other; they see, hear, feel and smell. Under this aspect, a system for human-computer interaction with the aim to make communication natural and intuitive has to apply or imitate this way of communication when interacting with humans. For the direction from the computer to the human, this means the human senses are stimulated for information transfer. The most applied senses today are sight (display) and hearing (sound or speech feedback), but the technological progress within the last few years has opened new opportunities to design communication in advanced ways, e.g., by HUD displays or virtual reality glasses (Figure 2.3), or even by addressing new senses.

On the input side, a computer perceives information from the user with sensors and input devices. Nowadays, the classic input devices keyboard and mouse are extended or replaced by new technologies like touch screens, speech input and gesture recognition. These new technologies, like multi-touch screens, are also continuously extended with new functionalities. The new iPhone 6s does an exemplarily job at introducing 3D touch and thus is capable of sensing how much pressure is applied to a display. Many

Figure 2.3 – Google Glasses and Oculus Rift

Sense	**Modality**	**Sense Organ**	**Input Device/Sensor**	**Output Device**
Sight	Visual	Eyes	Camera	Display
Hearing	Auditory	Ears	Microphone	Speaker
Touch	Tactile	Skin	Touch Screen	Vibration Alarm
				Force Feedback
				Braille
Taste	Gustatory	Tongue	Taste Sensor	-
Smell	Olfactory	Nose	Electronic Nose	Olfactory Display

Table 2.1 – The five senses and examples for corresponding sensors and devices

input devices can be considered to correspond to one human sense: cameras (sight), haptic sensors (touch), microphones (hearing) or even olfactory sensors (smell and taste) (Jaimes and Sebe, 2007). Table 2.1 gives an overview of the five human senses and examples of corresponding devices for input and output, including devices that are still uncommon today like the taste sensor (Tahara et al., 2011), electronic nose (Gutiérrez and Horrillo, 2014) and olfactory display (Matsukura et al., 2013).

The modality that is used to perceive interaction can differ between humans and computers. Humans recognise a body gesture by observing the movements. A corresponding approach for the computer that also uses the visual modality would be the interpretation of two-dimensional images from a conventional camera or three-dimensional information from a range camera like the Kinect from Microsoft. Additionally, a computer can access other sensor information that applies various modalities. In the case of the body movements, gesture recognition with accelerometers (Neßelrath and Alexandersson, 2009) shows that hand movements can be perceived by observing features that are hidden to humans, in the example the acceleration data of the hand instead of visual observations. Currently some sensors find no equivalent human sense like a Brain-Computer Interface (BCI) or biometric sensors, e.g., for measuring the galvanic skin response. They can extend the interaction concepts already known from the communication between humans with completely innovative possibilities.

In Human-Computer Interaction (HCI) a distinction can be made between intrusive and non-intrusive devices. While intrusive devices, like sensor gloves or virtual reality glasses, allow one to measure and present a larger set of information, they have to be attached first and may consequently restrict the free movement and sight of the carrier. In the industry context this is finally arguable, since the device may be part of the professional clothing. In a private context, e.g. in the car or a museum, this is hardly acceptable by the user. Here HCI systems have to rely on non-intrusive technologies like cameras with the disadvantage that normally the interpretation of the measured values is based on machine-learning and pattern-matching technologies, which can be more error-prone than intrusive devices, especially when considering external interferences like poor light conditions or noisy environments. With new wearable technologies like the smart watch, a first step has been taken in order to integrate more technologies into the people's lives without being intrusive.

2.2 Multimodal Human-Computer Interaction

Many surveys have been written about Multimodal User Interfaces (MUIs). Some of them give a general overview of the key aspects, concepts and frameworks in the evolution of multimodal interaction research (Dumas et al., 2009; Turk, 2014; Oviatt, 2012; Karray et al., 2008; Benoit et al., 2000). Others approach this topic from a certain perspective, e.g., from the multimodal fusion (Lalanne et al., 2009; Atrey et al., 2010) or computer vision (Turk and Kölsch, 2003; Jaimes and Sebe, 2007).

In the literature, terms related to this topic like *channel*, *medium*, *mode*, *modality*, *device*, *sensor*, *multimedia*, and *multimodality* are often used with diverse meanings and with time, the definitions of them have blurred. Bernsen (1997), e.g., states that in modality theory a medium is the physical realisation of some particular presentation of information, so it is more related to the human sensory modality, i.e., visual, acoustics and haptics. However, the term medium is too coarse-grained for a comprehensive classification of output variations. For instance, graphical displays allow different concepts for the presentation of information. It can be natural language text on the screen, symbols, animations or even virtual characters that communicate with the user using gestures. Acoustic output presents information in the form of alarm beeps, music or a synthetic spoken language. Thus, Bernsen defines the modality as "a mode or way of representing information to humans or machines in a physically realised inter subjective form, such as in one of the media of graphics, acoustics and haptics. Thus, a modality is a representational modality and not a sensory modality as the term 'modality' has traditionally been used by psychologists."

This example shows that the perspective of a psychologist significantly differs from a technical point of view. In order to have a clear foundation, the following definitions specify how the terms are used in this technically oriented thesis:

2.2.1 Terminology

Modality

In the literature like in Wahlster (2006b), the term *modality* often refers to the human senses employed to process incoming information: vision, audition, olfaction, touch, and taste. Other literature about MUIs like Oviatt and Cohen (2015b) from a more technical view use the term *modality* to describe the type of interfaces that are applied for user input -such as speech, pen, touch and multi-touch, gestures, gaze, and virtual keyboard- and system output -such as speech synthesis, Graphical User Interface (GUI) or sound.

Wasinger (2006) points out that with the term modality, a focus is set on the perception of the senses and the process in which user input is captured by the system or presented to the user. This definition combines both above mentioned definitions quite well and is accepted for this work.

Definition 1 (Modality)
Modality describes the perception of the senses and the process in which the user input is captured by the system or system output is presented to the user.

Code

With the term ‘code’, Maybury and Wahlster (1998a) refer to a system of symbols or information encoding (e.g., text, gestures or sign language). For the realisation of the code, a particular combination of user ability and device capability may be utilised. Thus, the modality actually used to present a code can differ, e.g., text can be entered via a keyboard or displayed on a monitor for output. Here the concrete involved modality is not crucial for the code; alternative examples for the concrete presentation of text are the spoken language or the braille language.

Definition 2 (Code)
A system of symbols or information encoding like text, gestures, or sign language. One code can be presented by diverse modalities.

Device

Nowadays, a device is not restricted to support only one specific modality or code. Moreover, modern devices can contain a wide range of different user interfaces. For example, current smartphones present information with sound, spoken language, GUIs, vibrations, and more. On the input side the user, e.g, interacts through multi-touch, speech input or motion, which is detected by accelerometers.

From the technical point of view of the Situation Adaptive Multimodal Dialogue Platform (SiAM-dp), presented in this thesis, a device is considered to be one unit in the environment that is connected to a multimodal dialogue application. Later in Section 2.4, in the context of CPEs, this unit will also be called a *cyber physical unit*. The various interfaces supported by one unit are called *device services* in this thesis. This term is examined in detail in the following definition and includes also sensors and actuators.

Definition 3 (Device)
One unit in the environment that is connected to a multimodal dialogue application. A device can combine several user interfaces for input and output but also sensors and actuators.

Device Service

Oviatt and Cohen (2015b) make a distinction between active input modes and passive input modes. They define active input modes as input modes “that are deployed by the user intentionally as explicit input to a computer system (e.g., speaking, writing, typing, gesturing, pointing)”. Equivalently, in this thesis active output modes are directly deployed to the user by a user interface (e.g., graphical output on the screen or speech synthesis). Since the initiative is originated from the computer system, it is difficult to speak of intentional and unintentional output. Thus, we consider output to be active output if there exists a possibly active equivalent in the interaction between humans.

Passive input modes are defined as “naturally occurring user behaviour or actions that are recognised and processed by the system (e.g., facial expressions, gaze, physiological or brain wave patterns, sensor input such as location). They involve user or contextual input that is unobtrusively and passively monitored, without requiring any explicit user command to a computer” (Oviatt and Cohen, 2015b). Passive input is typically recognised by sensors, like cameras, eye-tracker, time-of-flight cameras, and motion sensors. For system output, passive output modes are, e.g., realised with virtual characters and their facial expressions. Another set of devices that can be used for passive output are actuators. Exemplarily, lamps can be used to unconsciously attract the user’s attention on a specific region or object in a product shelf.

The generic term for an interaction mode that is provided by a device is the device service. The interaction mode in this context is the combination of modality and code that can be processed by the device service. Chapter 5 will introduce a more exact classification of the above mentioned types of device services.

Definition 4 (Device Service)
An interaction mode that is supported by a device. The interaction mode is the combination of modality and code that can be processed by the device service. This can be user interfaces for input and output, but also actuators and sensors. One device may contain more than one device services.

Definition 5 (Device Component)
The physical component of a device that is used to generate a system output or perceive a user input.

Table 2.4 shows the relations between the above mentioned terms using the example of a modern smartphone. Four typical device services are listed: speech synthesis, speech recognition, GUI, and hand gesture recognition. Every device service employs an individual device component. The modalities and codes also differ with the exception of speech synthesis and speech recognition. Here the decisive difference is the direction of the communication. Speech synthesis is used to present system output whereas speech recognition recognises user input.

Device	Device Service	Device Component	Modality	Code
	Speech Synthesis	Speaker	Speech	Natural Language
	Speech Recognition	Microphone	Speech	Natural Language
	GUI	Touchscreen	GUI / Multi-Touch	Text, Icons, Diagrams... / Gestures
	Hand Gesture Recognition	Accelerometers	Hand gesture	Sign Language

Figure 2.4 – Relations of the terms *Device*, *Device Service*, *Device Component*, *Modality*, and *Code*

2.2.2 Multimodal Systems

Nigay and Coutaz (1993) define the term *multimodal system* in the general sense as a system that "supports communication with the user through different modalities such as voice, gesture, and typing." They claim that the two main features of a *multimodal system* are the concurrency of processing and the combination of input/output data. Literally, "multi" refers to *more than one* and the term "modal" covers the notion of "modality". In their view multimodal systems, in contrast to multimedia systems, work on a higher abstraction level and understand the semantics of what they capture or present. So the multimodal system strives for meaning by also considering the context of an interaction.

In the W3C-specification EMMA (Johnston et al., 2009) multimodal interaction is defined as "the means for a user to interact with an application using more than one mode of interaction, for instance, offering the user the choice of speaking or typing, or in some cases, allowing the user to provide a composite input involving multiple modes". This definition implicitly says that interaction need not necessarily be always multimodal; sometimes interaction can also be unimodal.

Oviatt and Cohen (2015b) write that multimodal systems process two or more modalities, such as speech, pen, touch, gestures, gaze, and head and body movements. The modalities may coexist together but can be used simultaneously or alternately. "The input may involve recognition-based technologies (e.g., speech, gesture), simpler discrete input (e.g., keyboard, touch), or sensor-based information. Some of these modalities may be capable of expressing semantically rich information and creating new content (e.g., speech, writing, keyboard), while others are limited to making discrete selections and controlling the system display (e.g., touching a URL to open it, pinching to shrink a visual display). They mostly focus on user input but the same definition can be transferred to multimodal output where several modalities are combined in order to present information to the user on distributed output devices.

Definition 6 (Multimodal System)
Multimodal systems process two or more modalities for input and output, such as speech, pen, touch, gestures, gaze, and head and body movements. The modalities may coexist together but can be used simultaneously or alternately.

Oviatt and Cohen (2015b) also argue that in future, richly expressive communication interfaces will not only incorporate multiple modalities but also multiple linguistic codes and multiple representation systems (Figure 2.5). They use the term *representation* synonymously with the above defined term *code*. In a longer-term direction, thus communication interfaces can be established that are more capable of supporting people's ability to communicate fluently as they think and work (Oviatt, 2012).

As we will see in Chapter 5, concepts for the integration of multiple modalities in SiAM-dp can easily be adapted to support multiple representation systems as well as multiple codes.

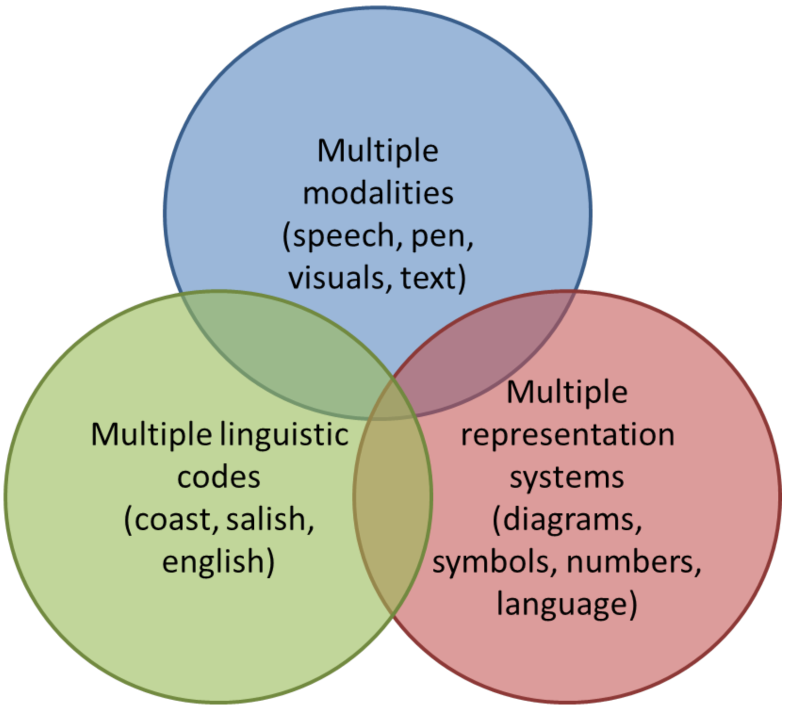

Multiple modalities (speech, pen, visuals, text)
Multiple linguistic codes (coast, salish, english)
Multiple representation systems (diagrams, symbols, numbers, language)

Figure 2.5 – Richly expressive communication interfaces support multiple modalities, representation systems, and linguistic codes (Oviatt (2012))

2.2.3 Advantages and Myths

Oviatt (2012) discusses the advantages of multimodal interfaces. In the following list some of them are outlined:

Flexible use of modes: Users can select their preferred input and output modes and combine or alternate between modes.

Accommodation of a broader range of users: Multimodal interfaces permit users of distinct age, skill level, native language status, cognitive styles, sensory impairment, and other temporary illnesses or permanent handicaps to interact with them.

Better adaptation to environmental conditions: Especially in a mobile scenario, multimodal interfaces can adapt to changing environmental conditions. For example, interaction is hands-free in an in-vehicle application or avoids speech in a noisy environment.

Improvement of efficiency: Studies showed that multimodal interfaces can improve efficiency. Efficiency is estimated by measuring the task completion time when using multimodal interfaces. This speed advantage also includes the time needed to correct recognition errors. Cohen et al. (2015) compared the task completion time of keyboard-based graphical interfaces with multimodal speech and pen ones and found that multimodal interfaces sped up the creation of complex simulation scenarios on a map by a factor of between 2.4 and 4-fold.

Facilitation of error recovery: Users can choose the input mode that they judge to be less error prone for a particular content.

Support of mutual disambiguation for input signals: Unimodal recognition errors can be recovered by semantic information from each input mode that supplies partial disambiguation of another mode.

Reduction of cognitive load: By distributing information across multiple modalities separate parts of the working memory are loaded (based on Baddeley's theory of working memory (Baddeley, 2012)). Thus, the user's overall cognitive load is minimized and their task performance enhanced.

On the other hand, Oviatt (1999b) disproves some myths about multimodal interaction. Some of them are briefly presented here:

Myth #1: If you build a multimodal system, users will interact multimodally - Mostly interaction is a combination of unimodal and multimodal contributions.

Myth #2: Speech and pointing is the dominant multimodal integration pattern - This myth represents the idea of the early multimodal systems that used a mouse-oriented metaphor. Nowadays many other modalities like gestures, gaze and touch can be part of multimodal interaction.

Myth #3: Multimodal input involves simultaneous signals - Input can also occur sequentially (see also section 2.2.4).

Myth #6: Multimodal integration involves redundancy of content between modes - More often multimodal content appears complementarily than redundantly.

Myth #7: Individual error-prone recognition technologies combine multimodally to produce even greater unreliability - In practice users figure out how to use the available input modes in an efficient way. Furthermore, mutual disambiguation of signals improves robustness.

Myth #9: Different input modes are capable of transmitting comparable content - Different modes differ in their functionality, the way they are integrated and the type of content they can provide.

Myth #10: Enhanced efficiency is the main advantage of multimodal systems - This is not always the case. Other advantages have already been presented above.

2.2.4 Integration and Fusion of Multimodal Input

Johnston et al. (2009) define *multimodal integration* as the process of combining input from different modes to create an interpretation of composite input. A synonym is the term *multimodal fusion* which is adopted from the terminology in physics. Multimodal fusion is a process that combines manifold types of input data, each associated with a particular modality. It is a fundamental task in the integration of various modalities.

Classification of Multimodal Input

Nigay and Coutaz (1993) call the absence of fusion *Independent Modalities* and the presence *Combined Modalities.* Serrano and Nigay (2009) organise the combination space of interaction modalities into two dimensions, the type of the relationship between modalities and the temporal relationship. The type of relationship is explained with the CARE properties Coutaz et al. (1995):

The CARE properties (**C**omplementary, **A**ssignment, **R**edundancy, and **E**quivalence) characterise multimodal interaction from the usability perspective on HCI. They are a set of properties that describe the relationship between modalities for reaching a goal or the next state in a multimodal system.

Equivalence - Expresses the concept of *free choice of modality.* Multiple modalities can reach the same goal and it is sufficient to use only one of them without any temporal constraint on them.

Assignment - Expresses the absence of choice. One, and only one, modality can be used in order to reach a goal. An example is the steering wheel of a car.

Redundancy - Two modalities have the same expressive power but are both required to be used within a temporal window in order to reach a goal. Redundancy can be important for safety relevant functionalities.

Complementary - Two modalities are used within a temporal window for reaching a goal. Both modalities are needed to describe the desired meaning. A speak-and-point system is a classic example of this.

Equivalence and Assignment are independent modalities and can be interpreted individually from one another. Redundancy and Complementary are combined modalities and require a multimodal fusion of the input. While redundant input must be compared and verified for an identical meaning, the complementary input must be combined in order to express the meaning.

Temporal Relationship and Synchronisation

As mentioned above, the time frame during which multimodal input occurs is relevant for the multimodal fusion. This implies the importance of the temporal synchronisation of all input devices. Vernier and Nigay (2001) specify five distinct combination schemes for the temporal relation between multimodal inputs (Figure 2.6). Three of the relations describe multimodal inputs that overlap and occur simultaneously (*Concomitance*, *Coincidence*, *Parallelism*). *Anachronism* and *Sequence* are sequential and are distinguished by the size of the temporal window between the usage of the two modalities.

The temporal relations can provide relevant indications of whether and how multimodal input should be combined. Early multimodal systems like the “Put-That-There” system by Bolt (1980) relied on the fact that multimodal constructions temporally co-occur.

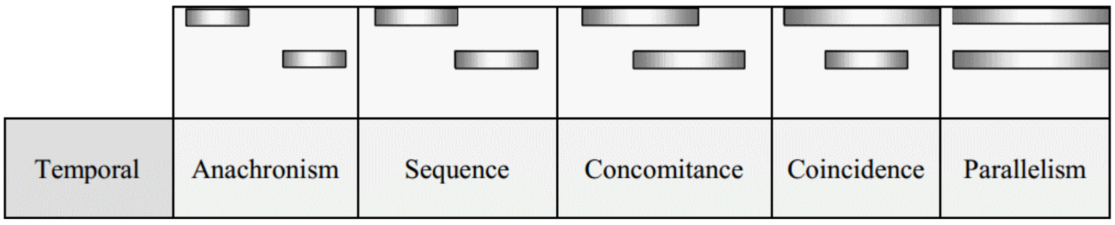


Figure 2.6 – Temporal relations between modalities (c.f. Vernier and Nigay (2001))

The meaning of the deictic term “that” in the spoken utterance “put that there”, e.g., was resolved with the object at which the user was pointing when it was spoken.

This multimodal integration approach seems to be suitable for multimodal speak-and-point systems but has a restricted practical use in the design of future multimodal systems that involve other modes like gestures or body movements without deictic-point relations (Oviatt, 2012). It turned out that the temporal overlap of signals not urgently determines which signals should be combined. A series of studies showed that there exist two distinct types of users with respect to integration patterns and that their integration patterns occur across the lifespan from children through the elderly (Xiao et al., 2002, 2003). An integration pattern here specifies the strategy of how users combine multimodal input with respect to the temporal relation. Simultaneous integrators overlap their input temporally, whereas sequential integrators begin with one mode after the other one has been finished (Oviatt, 1999b; Oviatt et al., 2005). Since a user’s habitual integration pattern remains highly consistent during a session, this may allow systems to automatically detect and adapt to a user’s dominant multimodal integration pattern. This may also include the temporal thresholds during the sequential use of modalities.

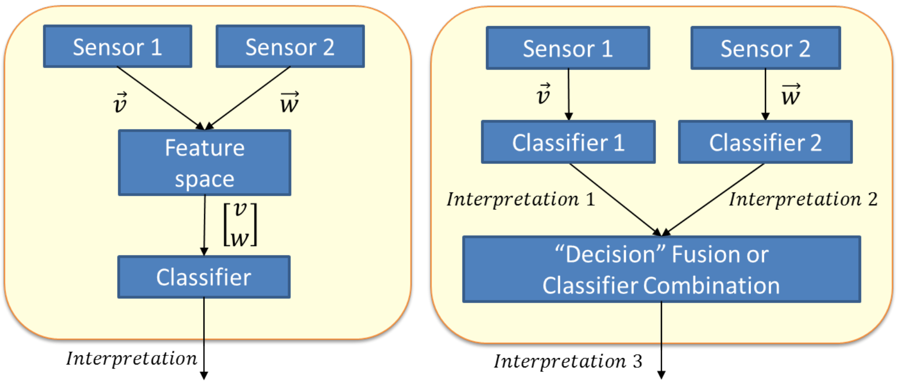
Fusion level

An important aspect for multimodal fusion is the appropriate fusion technique that is applied to combine incoming unimodal events into a single representation of the user’s intention. Literature often distinguishes between two stages where fusion occurs: *early fusion* and *late fusion* (Turk and Kölsch, 2003; Jaimes and Sebe, 2007; Nigay and Coutaz, 1993). The decisive factor here is the level of abstraction at which the fusion takes place.

Early fusion occurs at a *feature level.* The input signals are concatenated and provided to a joint classifier that generates an interpretation (see Figure 2.7a). The interpretation (or classification) is mostly based on machine-learning technologies like neural networks, or hidden Markov models. A classic example for early fusion is the audio-visual combination of speech and lip movements. Here the motion data from the lips are concatenated with features from the recorded voice in order to recognise a spoken utterance (Tamura et al., 2004).

During the *late fusion* or *decision fusion*, the signals are first classified independently on a feature level. After that the results are combined to a joint interpretation (see Figure 2.7b). The late fusion is realised on a semantic level and techniques like unification on graphs or Bayesian networks are employed in order to combine information.

Atrey et al. (2010) mention several advantages of the late over the early fusion. One is that the interpretations at a semantic level have the same form making their fusion easier. A second one is that for each single modality, the most suitable methods for analyzing the input data can be applied, making the process more flexible than the early



(a) Early fusion at a feature level **(b)** Late fusion at a semantic level

Figure 2.7 – Multimodal fusion on distinct levels (Oviatt and Cohen, 2015b)

fusion. Wahlster (2003) explains that at a semantic level, the back-tracking and reinterpretation of a result is easier. Furthermore, the development process is less complex since multimodality with new modalities can be handled without specifying all varieties of cross-modal references in advance. Oviatt and Cohen (2015b) point out that the development process is simplified because commercial "black box" recognisers can also be applied, which provide no access to their internal state or data. Late fusion is able to fuse modalities that are not time-synchronous. With early fusion, the feature vectors usually have a close temporally bound. However, early fusion can gain potentially useful information that would already have been thrown away when the late fusion is applied (Wasinger, 2006).

In SiAM-dp the focus is laid upon the late fusion process, since the system works with already semantically represented content that is provided by modality-specific interpreters/recognisers. Nevertheless, it is possible that recognisers already applied early fusion for the interpretation of input from multiple modalities before they provide the result to the dialogue system.

2.2.5 Presentation Planning and Multimodal Fission

The previous section discussed the fusion of modalities during input; in this section we look at the output side. *Modality fission* is the process of splitting semantic representation from an intended modality-free output into a multimodal presentation to be realised. Like the term *fusion*, the term *fission* has also been borrowed from physics and emphasises that the output is split into several output modality presentations. Modality fission includes the decision of how the output is channeled and coordinated throughout the diverse available output modalities based on the user's perceptual abilities and preferences (Costa and Duarte, 2013).

Foster (2002) subdivides the tasks of a multimodal fission component into three parts:

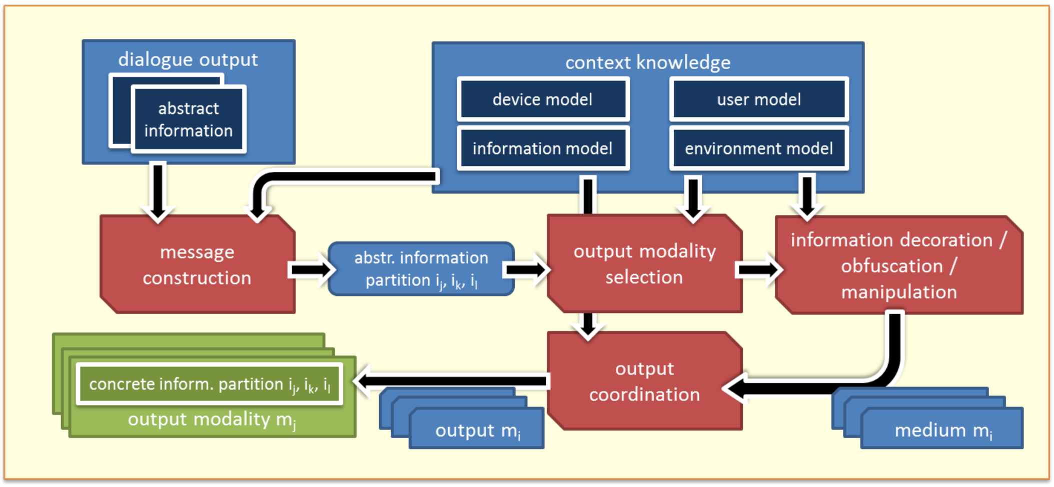


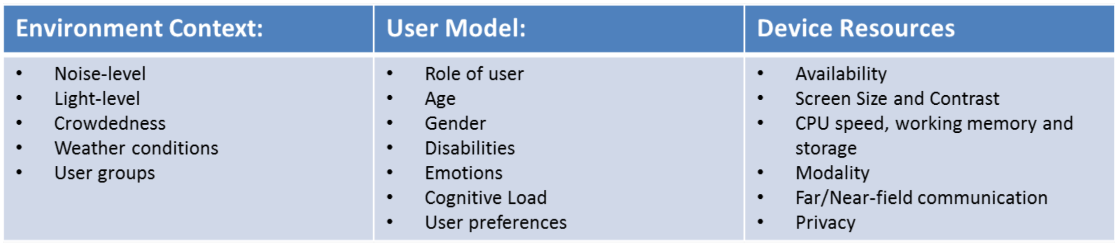
Figure 2.8 – The multimodal fission process for the creation of context aware output from abstract information

Content selection and structuring - Often the content to present is already selected by the dialogue management component and provided on a semantic level. In the first step, a fission component must divide the overall meaning into elementary elements that can be presented to the user. Here the main approaches are schema-based or plan-based.

Modality selection - In the next step the devices and modalities are selected that contribute to the multimodal output. The available devices are described by several features that include the type of information they can handle, the perceptual task they permit, the availability, the characteristic of information to present, resource limitations and the user's profile which includes his abilities, skills and impairments.

Output Coordination - After the output modalities are selected, the modality specific realisations must be created. For these modality attributes, spatial and temporal parameters, as well as user characteristics, etc., are considered. Especially for multimodal outputs with cross-modal references, the scheduling, synchronisation, and coordination of presentations play a large role.

Honold et al. (2012) implement this concept in an adaptive probabilistic approach for multimodal fission. Their fission process is depicted in Figure 2.8. Outgoing from a modality-independent dialogue output, they first partition the data items into elementary data. Then they use a probabilistic reasoning approach for the selection of the devices and modalities to involve. In the next step they post-process the output data and, e.g., obfuscate private messages if only public devices for presentation are available. In the last intermediate step, the final concrete output realisations are created for each information item. Finally, the output is distributed between the target output devices.



Environment Context:	User Model:	Device Resources
• Noise-level	• Role of user	• Availability
• Light-level	• Age	• Screen Size and Contrast
• Crowdedness	• Gender	• CPU speed, working memory and storage
• Weather conditions	• Disabilities	• Modality
• User groups	• Emotions	• Far/Near-field communication
	• Cognitive Load	• Privacy
	• User preferences	

Table 2.2 – Features with effect on presentation output planning

All the processing steps involve context knowledge about available devices, users, and the environment in their decisions. Wasinger et al. (2003) present several features from diverse context sources that affect on the presentation output planning. They are summarised in Table 2.2 and are extended with some features that play an increasing role in CPEs. Endres (2012a,b) presents with PresTK a platform for the situation-aware presentation of messages and infotainment content for drivers. The main goal of the platform is to present a dramatically increasing number of in-car information systems to the driver without also increasing the risk of driver distraction. In the first step, the system applies techniques from scheduling and presentation planning in order to avoid conflicts when competing for scarce resources such as screen space. In the second step, the system considers the cognitive capacity of the driver.

Wasinger (2006) also sets a focus on *symmetric multimodality* that has been introduced by Wahlster (2003) within the SmartKom project. The main statement here is that all input modes should also be available for output, and vice versa: "only true multimodal dialogue systems create a natural experience for the user in the form of daily human-to-human communication, by allowing both the user and the system to combine the same spectrum of modalities". Thus, the presentation planner must take into account the modality a user applied for input, and adequately adapt the selection of output modalities. For example, speech input is responded to with speech output, interaction with a GUI is responded to with GUI updates and if a user performs a pointing gesture in order to identify an entity in the room, the reaction of the system could use light spots in order to highlight the referred entity or maybe even use a robot arm.

One important design principle in the SmartKom project (Wahlster, 2006b) was "no presentation without representation". This means that the generated multimodal presentations must be explicitly represented in order to ensure the dialogue coherence in multimodal communication. This plays a relevant role for the resolution of anaphoric, cross-modal, and gestural references of the user. In the developed system, a text generator provided a list of referential items that were mentioned in the last turn of the system. A display management component permanently kept track of the currently presented screen content in an internally managed model for the display context.

2.3 Dialogue Systems

A dialogue system is a software agent that allows users to converse with a computer in a coherent structure. Its origins are in spoken dialogue but over time many additional modalities have been employed in multimodal dialogue systems like text, speech, haptics, graphics, gestures, and other modes. Figure 2.9 shows an abstract reference architecture for multimodal dialogue systems introduced by Bunt et al. (2005) that extends the original reference architecture of Maybury and Wahlster (1998b). The typical architecture comprises three sequential processing phases.

1. Analyzing and understanding user input. This includes modality specific recognisers and analysers, and fusion as well as discourse processing.
2. Dialogue management and action planning.
3. Planning and generation of system output by modality fission and modality specific generation and realisation.

Furthermore, the architecture includes models for user, discourse, context, domain, task, media, and application that are accessible by all components.

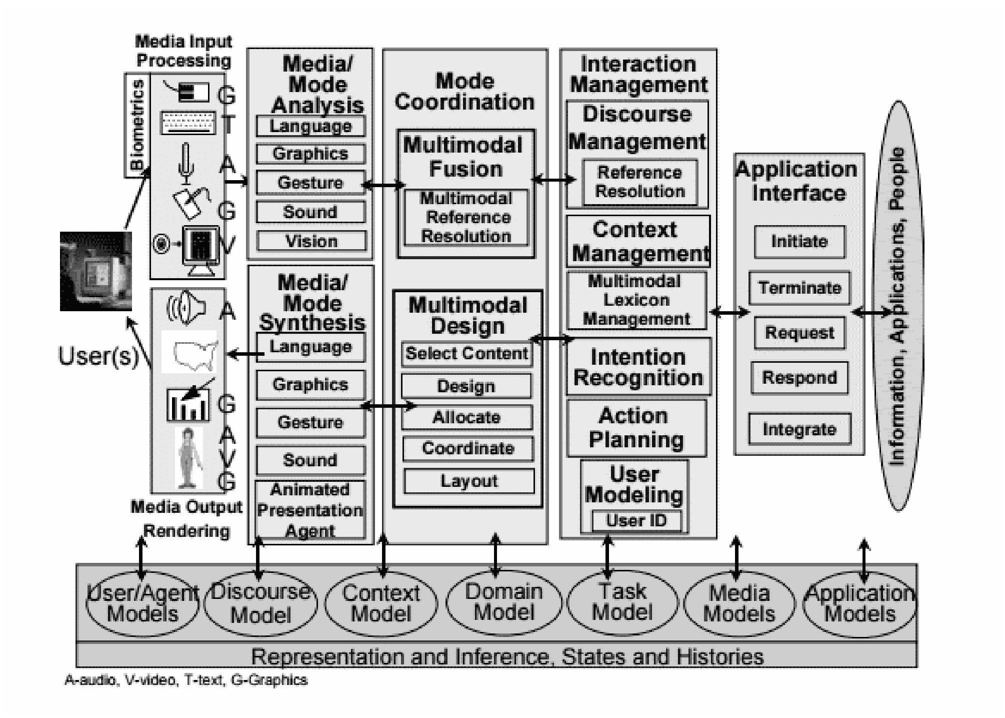


Figure 2.9 – Reference architecture for multimodal dialogue systems (c.f. Bunt et al. (2005))

2.3.1 Dialogue Management

The dialogue manager is the component in a dialogue system which controls the architecture and structure of the dialogue. Traum and Larsson (2003) define the following functions as the main tasks of a dialogue manager:

- Updating the dialogue context on the basis of interpreted communication. The communication can originate from the human user, the system itself, or any other connected software agent.
- Providing context dependent expectations for interpretation of observed signals as communicative behaviour.
- Interfacing with task/domain processing (e.g., database, planner, execution module, other back-end system), to coordinate dialogue and non-dialogue behaviour and reasoning.
- Deciding the content that is expressed next and when to express it.

While the first two points concern the management of the dialogue context and the context-based interpretation of communication, the latter two rather handle the control of the conversation with the user. A distinction is made between *user-initiative*, *system-initiative*, and *mixed-initiative* systems, whereby the initiative belongs to the speaker in control of the conversation (Jurafsky and Martin, 2009, page 865). *User-initiative* systems are typical command and control systems. In a *system-initiative* system, the conversation is completely controlled by the system. Thus, the system asks a question to the user and solely reacts on inputs of the user that exactly answer the question. An improvement of this concept is *universal* commands that can be said anywhere in the dialogue and for example provide the user a shortcut for the navigation to a help or a main menu.

Often a user wants to communicate something that is not exactly the answer to a specific question, maybe in a sentence that provides more than one relevant piece of information. In a restaurant-reservation example, this would be the sentence *"I want to reserve a table at five p.m. on Monday for three people"*. Here, besides the day and the time, the number of people for the reservation is also given. This type of interaction is more natural for humans. Systems that support such a shift of conversational initiative are called *mixed-initiative* systems.

Dialogue Management Approaches

Several approaches for dialogue management are mentioned in literature (Bui, 2006; Jurafsky and Martin, 2009):

Finite state-based - This approach is the simplest one and very well suited for less complex dialogue systems with well-structured tasks. However, for handling a mixed-initiative dialogue, a finite-state architecture is inappropriate due to the enormous number of states that would be required to handle each possible subset of questions and answers.

Frame-based - This approach is analogous to a form-filling task in which a predetermined set of information is collected. A frame-based dialogue manager collects information from the user by asking questions until enough information is available to perform a task. If a user happens to answer more than one question at a time, the system has to fill the appropriate frames.

Information state-based - This architecture consists of five components: Information state, dialogue act interpreter, dialogue act generator, a set of update rules, and a control structure that selects which update rules to apply. The term "information state" is quite abstract and might include things like the discourse context and the common ground of dialogue participants, their beliefs or intentions, user models, environment models, and so on. Thus, in contrast to a static state in the finite state-based approach, the information state is more complex and includes the values of many variables, the discourse context, and other elements. The update rules are responsible for modifying the information state based on the information of the dialogue acts. One subset of these rules is called *selection rules* and is used to generate dialogue acts in order to control the dialogue.

Plan-based Dialogue Agents - Plan-based approaches are based on the idea that people communicate in order to achieve goals, which includes the change of the mental state of the listener. Thus, plan-based models are often referred to as beliefs, desires, and intentions (BDI) models, which were first introduced by Perrault and Allen (1980) and Cohen and Perrault (1979). In the plan-based theory, the speaker's speech act is part of a plan and it is the listener's job to identify and respond to this plan. Communication and conversation are thus just special cases of rational actions in the world that can be planned as any other action by applying AI planning techniques like the TRIPS agent (Allen et al., 2001).

Recent approaches represent the underlying structure of a dialogue using **probabilistic models** (Lison, 2012). For this, rules are specified using high-level conditions and effects and are defined as structured mappings over variables of the dialogue state. Nowadays, probabilistic models such as Bayesian Networks are in widespread use in spoken dialogue systems, but their scalability to complex interaction domains remains a challenge. Probabilistic models should help to make dialogue system more robust against noise and uncertainty and to be capable of automatically learning and optimising from data, making them more flexible and adaptive.

2.3.2 Context Resolution

Often contributions in a dialogue have to be interpreted with respect to the actual context, which includes the context of the world, the actual discourse, but also coherent contributions of other input channels (compare, e.g., Bunt (2000)). Hence, in multimodal dialogue systems it is inevitable to incorporate contextual information in order to resolve linguistic phenomena like referring expressions.

Referring expressions

Referring expressions are a key linguistic phenomenon in verbal utterances for the identification of specific entities in the real world. The referred entity is called the *referent*. Pfleger (2007) gives an extensive introduction of the role and the various types of referring expressions from a linguistic point of view. *Deixis* or *deictic expressions* is a group of referring expressions that refer to "some entity or concept of the physical, situational or discourse context". They are immanently dependent on the contextual information and can only be interpreted if the context is considered. Since they play a relevant role in the representation of meaning in communicative acts, we give a short overview of some referring expressions Pfleger supported in his discourse resolution for multimodal dialogues.

Anaphora - The term anaphora stems from the Greek word meaning "carrying back" and is a reference to an entity of the preceding discourse. In linguistics, an anaphoric expression is a pronoun or a nominal phrase that is linked to a noun which has been previously introduced in an utterance.

Place Deixis - Describes a spatial reference either relative to the participants of a communicative act or to other entities in the context. The object with respect to which the figure is located is called *relatum*. The important thing for the resolution of a spatial deixis is the *frame of reference*. This can be intrinsic if the speaker takes the viewpoint of the relatum; relative, if the object is located relative to another object; or absolute, if an unambiguous reference point is used.

Time Deixis - Describes time points or time spans that are relative to the time point when a communicative act was produced. In natural language, adverbs like *then*, *now* or *tomorrow* express a temporal deixis. Temporal deixis also comprises complex compound temporal references like *next Monday* which consist of an adverb and a non-deictic name or unit of time that is modified.

Exophoric References - These are references to the visual or situational context of the discourse. This can, e.g., be an object in the physical environment or objects that are presented on a graphical display.

References to Collections - Sometimes an expression references an object in a collection of possible referents, e.g., in a list of entities. Here a differentiation criterion like *“the third entry”* describes the referred object.

Cross-modal references - In a multimodal contribution, sometimes the content of one input modality refers to content that is provided by a second modality. For example, in the combination of a pointing gesture with the utterance *“What is this building?”*, the pronoun refers to the entity that is indicated by the pointing gesture.

Ellipsis - In an elliptical construction, one or more words of an expression are omitted, e.g., if some of the constituents have already been mentioned in a previous turn. This especially may occur in information-seeking dialogues. The following example shows an elliptical construction:

User: *What is the menu for today?*
System: (Presents the actual menu)
User: *And for tomorrow?*

Constraints on References

Communicative acts that contain referring expressions can additionally provide restrictions on the referred object which is valuable information for the resolution of matching referents. Pfleger introduced two types of constraints:

Syntactic Constraints - A referring expression can contain linguistic information about number, person, and gender of the referent. Usually these features must match the result of the reference resolution.

Semantic Constraints - A referring expression can also contain semantic information about the referent. For example in the utterance *“turn on this lamp”*, it is semantically clear that the user refers to a lamp and not to, e.g., a ventilator in a room. Semantic constraints can provide information about the type but also about features of an object like in the utterance *“the green lamp”*.

2.4 Cyber-physical Environments

The combination of the physical environment, the virtual world, and data from local networks and the internet is called CPE and enables a new spectrum of applications and business models. A CPE is the integration of several connected Cyber-physical Systems (CPSs) of the environment.

MacDougall (2013) defines CPSs as “enabling technologies which bring the virtual and physical worlds together to create a truly networked world in which intelligent objects communicate and interact with each other”. Thus, they integrate computation with physical processes. CPSs are a complement to *embedded systems*, which are engineered systems which combine computing with physical processes. Examples for the latter are

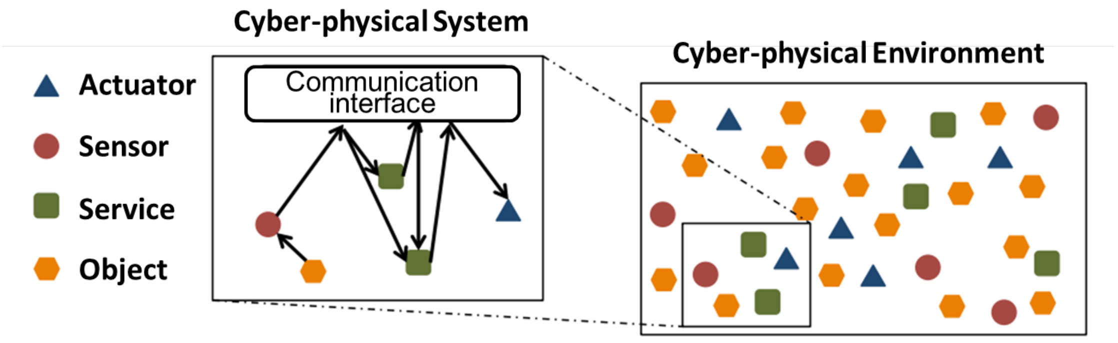


Figure 2.10 – A CPE is the integration of multiple CPSs (c.f. Kahl (2014))

automotive electronics, aircraft control systems, home appliances, etc. These systems all have one thing in common; they are realised in a closed box that does not connect its computing capability to the outer world. "A common feature of almost all CPS is that they heavily rely on networking" (Giese et al., 2011). Thus, together with the integration of modern multimodal human-computer interfaces and software-based internet services, this technology enables new interaction possibilities and applications that make the frontiers between the virtual and real world disappear. Furthermore, the connection of several embedded systems in the environment allows the provision of higher-level services.

Kahl (2014) gives an overview of the components in a CPS and the integration of multiple CPSs to a CPE. A CPS is a network of sensors, actuators, objects, and services (see Figure 2.10). Sensors perceive changes in the environment and can detect and monitor real objects and people. A set of dedicated services can process this sensor information and infer further findings about the actual physical context. Furthermore, a CPS is able to manipulate and interact with the environment by accessing actuators that form the counterparts of sensors. Then control commands in the form of electronic signals are transformed into physical measurable actions. The control of the actuators is a reaction to the processed sensor information. In a simple example, the signal of a motion sensor turns on the light in a room.

CPSs are an important requirement for the realisation of the *Internet of Things* (Wahlster, 2013). Together with the *Internet of Services* (Wahlster et al., 2014), they are key technologies for the future projects *Industrie 4.0* (Kagermann et al., 2013) and the *Smart Service Welt* (Acatech, 2014). Acatech (2011) see further potential of CPSs in energy (smart grid), networked mobility, health (tele-medicine and remote diagnosis), and industry.

One possible application scenario of CPEs is the *Smart Factory*, where workers are supported during their work with sensor and service information, e.g., for localizing tools and construction units, controlling devices, or synchronising dates and tasks. The applied user interfaces are wearables or interfaces that support the interaction from a distance. Thus, the worker is able to retrieve information without leaving his workplace, dropping his tools, or polluting the input device.

2.4.1 Human Computer Interaction in CPEs

The emergence of CPEs requires a complete rethinking of the interaction between the environment and the users that become part of the environment, move in it, and interact with it. The classical interaction concepts where a user exclusively interacts with one stationary device, PC, or smartphone become less dominant and make room for concepts where the user interacts with the environment (Human-Environment Interaction (HEI)). Input and output devices are combined with sensors and actuators in the environment for the communication between humans and computers, and thus new interaction paradigms will evolve. Novel interface technologies like speech interaction, gesture recognition, distributed displays, eye-tracking, and wearable devices reinforce the trend to carry the interaction away from stationary places where keyboards, mouse, buttons, and switches are the usual input devices, into the room.

On the output side, the system may use classic devices for interaction, like displays or speakers. Additionally, new actuators are available in order to gain the attention of the user, e.g., a light spot that is turned on. On the input side, the stimulus for a system's reaction can be triggered actively by the user, e.g., by pressing a button, or it can be an autonomous decision of the system based on received sensor information. However, if a user is aware of the trigger conditions for a system's reaction, it is possible that he triggers a sensor event on purpose. Thus, the transition between direct and indirect interaction may blur. In the perception of the human, the old-established interaction with only one assigned modality for one functionality changes to an interaction with the environment independent from the actually involved sensors and actuators.

2.4.2 Requirements

Often, higher standards are demanded from an embedded system compared to general-purpose computing systems. From household and consumer electronics, a reliable and robust functionality is expected. A crashing stove or TV is not tolerated by the customer. This seems even more essential for aircraft or automotive control systems. Here, a malfunction can decide between life and death. In CPEs a large number of heterogeneous embedded and physical subsystems are networked and have to interact concurrently as well as collaboratively. Meanwhile, the environmental conditions can permanently change and the CPE must be able to adapt to unpredictable situations and be robust enough in order to react to subsystem failures (Lee, 2008).

Kahl (2014) argues that the communication infrastructure of a CPE must be flexible and preferably error resistant. Especially the number of integrated components is not a priori known and can dynamically change. He expects that an event-based communication service like his Event Broadcasting Service (EBS) will be able to ad-hoc bind new sensors and actuators into a CPE. The concrete interactions can be seen as events at a specific time point. The involved actuators and sensors thereby interfere with each other. An asynchronous communication would help to avoid deadlocks.

2.5 Summary

This chapter introduced the fundamental concepts for this thesis. Therefore, first an overview of human verbal and nonverbal communication was given and examined how communication is realised in human-computer interaction. The following part gave a definition for multimodal systems, reflected on advantages and myths, and finally described the challenges that arise from multimodal integration. Afterwards a short introduction into dialogue systems, strategies for dialogue management, and the task of context resolution was presented. The last part explains the term “cyber-phyiscal environment”.

3 Related Work

During recent years, multimodal interaction has emerged as an important topic for the research in Human-Computer Interaction (HCI). New technologies and the use of mobile devices in various domains have increased the requirement for more flexible, more robust, and more natural interaction concepts. This chapter presents relevant work in the area of multimodal interaction from the beginning to the present and summarises projects and numerous use-cases that show the relevance of multimodal interaction in state-of-the-art research. Furthermore, the chapter outlines existing frameworks for the development of multimodal applications. The final sections present and compare several models for the representation of multimodal interaction. In the first part the focus is set on existing markup languages; the second part observes related work that deals with the semantic annotation of dialogue acts.

3.1 Overview of research in multimodal interaction

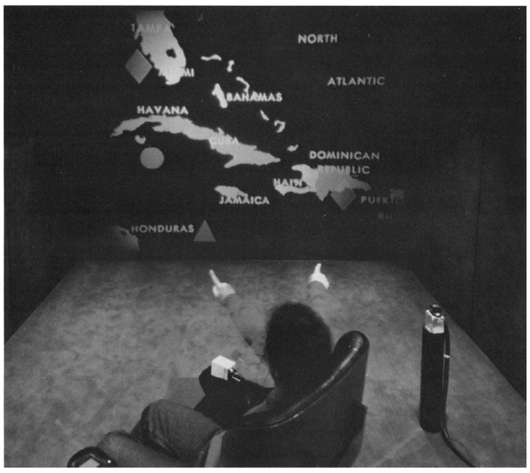
This section gives an overview of multimodal dialogue applications. It starts with the evolution of systems for multimodal interaction. In the first part it describes the breakthrough of the first multimodal systems which was the starting point for the development of a great number of systems that challenged a wide spectrum of integration issues. With the experience drawn from these systems, the first design rules were formulated. This included strategies for the fusion of modalities but also requirements for multimodal architectures. Since the already presented classical systems mostly integrated Graphical User Interfaces (GUIs), speech interaction and pointing gestures, the last two subsections deal with systems that extend the set of modalities with devices for nonverbal communication, gaze and gestures. They summarise possible areas of application with respect to the domain and their contribution to research in HCI.

3.1.1 Speech & Pointing

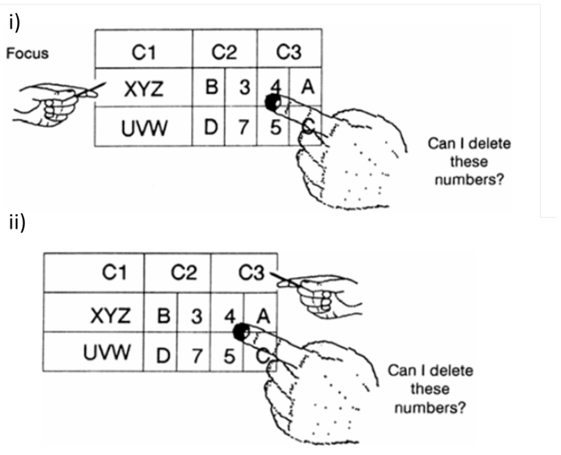
The *"Put That There"* Bolt (1980) system is one of the earliest multimodal concept demonstrations. Bolt built a media room with a wall-sized screen display and a user chair in front of it (see Figure 3.1a). The system enables the user to interact by voice and pointing gesture inputs in a spatial data management context. Commands like "create a blue square there", "Put that there" or "Make that smaller" allow the user to create, move or manipulate geometric objects on the screen. All of these phrases are incomplete since either the information about the object to manipulate or the target position of a movement action is missing. The missing content is complemented with the integration of pointing gestures that provide spatial information and allow one to resolve pronoun references and to eliminate ambiguity. Semantic processing is, e.g., realised by replacing the deictic term 'there' with the x,y coordinate indicated by the cursor at the time of the utterance.

The *CUBRICON* system from Neal et al. (1989) supports multimodal input and output in the context of map-based tactical mission-planning. It enables the user to interact using spoken or typed natural language in combination with pointing gestures generated by mouse input on a graphical display. In the other direction, the system multimodally presents information distributed to three output devices: Two displays and speech synthesis. Thus, it combines generated spoken natural language output with graphical pointing gestures. For example, if speech output provides information about an object, the icon that represents this object is simultaneously highlighted by blinking. The system uses a semantic based reference resolution process that supports the handling of ambiguous pointing gestures by considering the type or particular properties of the object that is represented by a selected icon. For example, if more than one icon is close to the coordinate of the pointing gesture, the utterance "What is the status of this <point> airbase?" is only combined with icons from objects of the type airbase.

The *XTRA* (Wahlster, 1991) system allows the user to combine natural language input together with pointing gestures in the context of an expert system that assists the user in filling out a tax form. The goal is to simulate a face-to-face conversation between humans where they frequently use deictic gestures parallel to verbal descriptions for referent identification. A focus lies on the interpretation of distinct pointing gesture granularities that range from exact pointing with a pencil, via standard pointing with the index finger, to vague pointing with the entire hand. Since each granularity level results in a different number of referential candidates, is it necessary to involve more knowledge in the reference resolution process. The content of the knowledge base embraces the tax form and the form hierarchy, the pointing gestures, a conceptual domain-specific model, the functional-semantic structure of natural-language input, and the dialogue memory. In a multi-step approach, the correct reference is resolved by analyzing the pointing gesture, the semantics of the verbal object descriptors, and the appearance of an object in the dialogue memory (Kobsa et al., 1986). Additionally, simultaneous pointing gestures with both hands are supported. Figure 3.1b shows how this can help



(a) The "Put That There" system (Bolt, 1980)



(b) Simultaneous pointing gestures in the XTRA system (Wahlster, 1992)

Figure 3.1 – Multimodal interfaces with combined spoken and gestural interaction

to prevent ambiguities. Here, the pencil in one hand specifies the focus by pointing to a region of the form, and the index finger of the other hand points to a specific object in the marked region. Although the finger of the second hand points at the same location, the selected numbers differ depending on the location of the pencil, which is used for focusing (i: {3,4},ii:{4,5}). Three reasons for the advantage of using pointing gestures are mentioned: The natural language dialogue is simplified by saving the speaker the generation, and the hearer the analysis of complex referential descriptions; they make reference possible in situations in which linguistic reference is not sufficient; and they allow the speaker to be imprecise or ambiguous, especially if the precise technological term is unknown to him.

The AlFresco system Stock et al. (1996) is a multimodal system that integrates natural language and hypermedia. It is an interactive system for users interested in frescoes and paintings and provides information, images, and videos of Fourteenth Century Italian frescoes and monuments. Besides an understanding of natural language, the system integrates the typing of sentences and navigating in underlying hypertexts using a touchscreen. For a better hypertextual exploration, the output of images and text with buttons offers new entry points for further communication. It has been one of the first systems that managed the coherence between dialogue and displayed output. The dialogue manager provides a graphical representation of the discourse which helps to limit the problem of opacity in the system's behaviour and thus allows the user to easily resolve misinterpretations. With the support of the resolution of anaphoras and deictic references on displayed images and hypertext buttons, the system allows much more effective access to information than a system with natural language only communication.

The *QUICKSET*-system (Cohen et al., 1997) is a multimodal pen and voice system

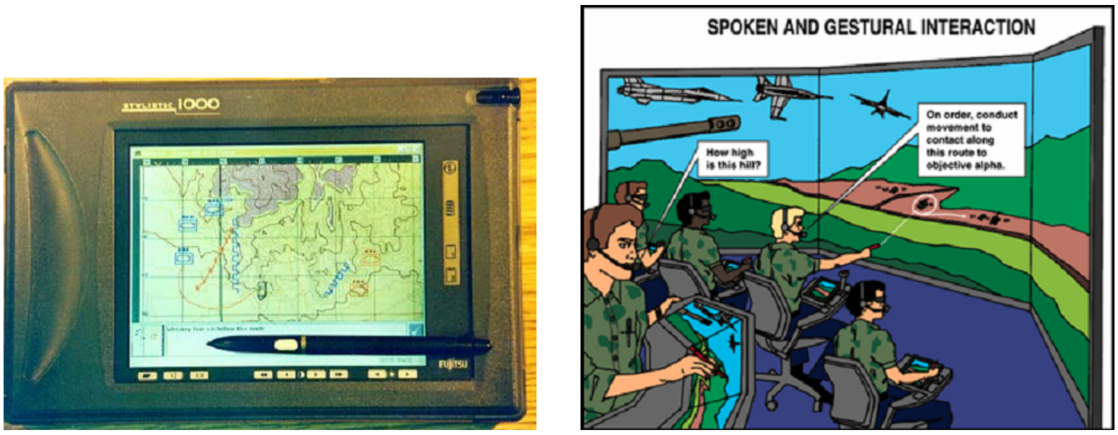

STYLISTIC 1000
FUJITSU
SPOKEN AND GESTURAL INTERACTION
How high is this hill?
On order, conduct movement to contact along this route to objective alpha.

Figure 3.2 – The collaborative and multimodal pen and voice system QUICKSET (Cohen et al., 1997)

that allows the collaborative interaction via a number of distributed devices. It provides a multimodal interface to various applications by integrating components responsible for speech recognition, natural language generation, graphical user interfaces and multimodal integration. The architecture allows the connection to distributed devices and the outsourcing of expensive input processing to resource-rich devices. Communication is achieved via WLAN and through a distributed multi-agent architecture. The same interaction capabilities can be enabled for distinct types of supported devices, e.g., handhelds, desktops, and wall-sized terminals. The system allows one to realise applications in diverse scenarios with a special focus on map-based interaction. A core functionality is the unimodal and multimodal integration of spoken language and pen input. Whereas speech input is the main modality for initiating interactions with the system, the pen input can provide valuable additional input information. The pen input is interpreted by a gesture recognition agent that uses neural network and hidden Markov models and is able to recognise 68 pen-gestures, including various military map symbols (platoon, mortar, fortified line, etc.), editing gestures (deletion, grouping), route indications, area indications, taps, etc. Furthermore, hand-written text can be interpreted.

One presented example application (Figure 3.2) is a military strategy simulator. The user can use pointing gestures combined with utterances to create new objects like tanks at a specific point on the map. He can also add barbed-wire fences or fortified lines by drawing lines at the desired locations. The typification is done either unimodally by drawing the appropriate military symbol or multimodally by saying the label.

The author specifies some multimodal architecture requirements for future human-computer interfaces that mean a further evolution step and additional value over the multimodal systems of the first phase. First is a flexible asynchronous architecture that allows multiprocessing and parallel running recognisers and interpreters. Their result should be a set of time-stamped meaning fragments for each input that are described in a common representation. With the support of a time-sensitive grouping process and a fusion concept, it should be possible to semantically combine meaning fragments from each modality stream to a joint interpretation.

The QUICKSET system fulfills these requirements with a flexible asynchronous framework and employs continuous speech and continuous gesture recognisers running in parallel. The representation of meaning is solved with typed feature structures that are very well suited to multimodal integration because they allow the sharing of structures and the representation of partial meaning. By applying typed feature structure unification on the input, it is possible to combine complementary and redundant information whereas contradictory information can be recognised and refused.

Further early works on the combination of spoken and gestural interaction are presented in (Siroux et al., 1995; Cohen et al., 1997; Oviatt, 1996).

3.1.2 Gaze, eye and head-tracking

While the previously mentioned modes of interaction primarily focused on deictic gestures that are performed either directly by hand or with the help of a pointing device, other research approaches used eye-tracking for the resolution of deictic references. It can be assumed that eyes are an ideal pointer and a person's eye movements and eye fixations strongly correlate with the person's attention to an object in the environment (Just and Carpenter, 1976; Starker and Bolt, 1990). Sibert and Jacob (2000) argue that people easily gaze at the world while performing other tasks. So the additional effort for eye-gaze combined with other input techniques is quite low. In their studies they found out that the selection of objects by eye-gaze is faster than selecting with a mouse.

People tend to look at things that are in the focus of their interest. Thus, it can be expected that a user is tracking the object he is talking about while he is formulating a speech command. Koons et al. (1991) utilise this feature of gaze and augment the interpretation of deictic references when the input from other modes is partial or segmented. They designed a prototype system that collects input from speech, gestures, and eye movements. In their emergency scenario they display a two-dimensional map on the screen and show icons representing helicopters, airplanes, trucks, fire crews, and fire locations (Figure 3.3). With the multimodal interface, the user is able to request information or give commands to modify the contents of the map database. The input from all three modalities is interpreted by modality-specific parsers that represent their results in a common intermediate frame-based form. The important thing is that each syntactical token of the interpretation is annotated with timing information. Since in their prototype both gestures and eye movements are adequately treated as contributors with deictic interpretations, the timing information can be used to replace ambiguous or underspecified information of the spoken request with their interpretation results. The resolution of the missing content is based on typed features of a taxonomy for the presented objects.

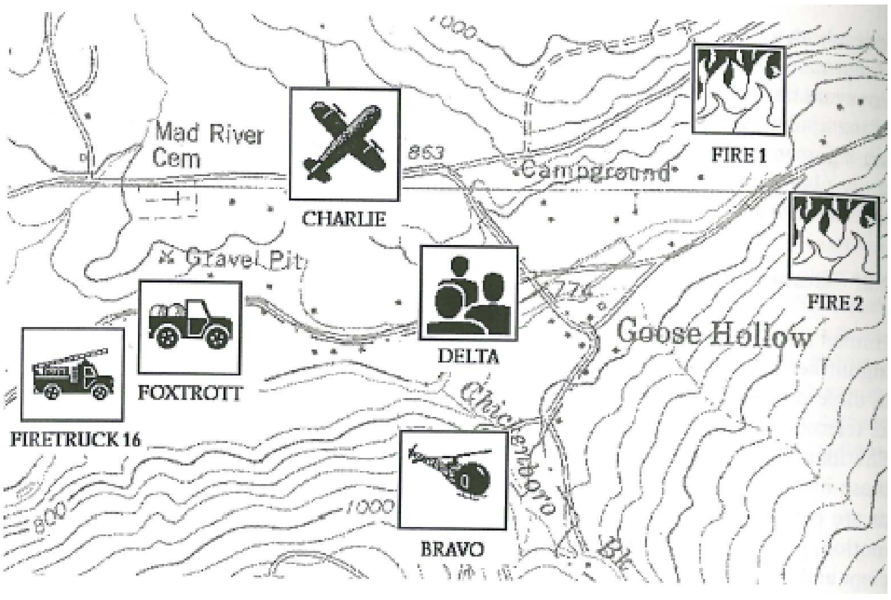


Figure 3.3 – Multimodal interface for speech, gesture and eye-gaze input. (Koons et al., 1991)

Moniri et al. (2012) extend the concept of eye-gaze based reference resolution with the identification of real objects in the environment. Their scenario takes place in a modern car which is instrumented with an eye tracker, head tracker, GPS logging module, two displays, and a speech recogniser. The aim of the presented system is to improve the car infotainment system with multimodal communication. While the driver controls the car, he is able to request information about objects in the environment. For example, the command "what is this building?" that is recognised by the speech recogniser is answered with information about the building that is actually in the line of sight of the driver. In contrast to the system introduced before, the referenced object is not displayed on a screen where the current 2-dimensional location is known but it is part of the real 3D world. Thus, the spatial resolution of the object is much more complex and is solved using an algorithm that regards the eye-tracking data, head tracking information, GPS position, orientation of the car, and a spatial model of the environment.

In Kern et al. (2010), drivers' glances at the screen are used for an explicit gaze-based interaction. The approach realises a direct interaction on a visual display without the drawback of taking the hands off the steering wheel. The benefit of a hands-free interaction is an increase in road safety. The idea is to replace all actions that can be performed by a single touch on the touch screen with gazes. The duration of the gazes on the screen should not significantly increase in comparison to the use of the touch screen. While looking on the screen, the user gets a visual feedback by highlighting the object the user is currently looking at. Two strategies are tested for the selection of an item. The first strategy uses a gaze in combination with pressing a button. The

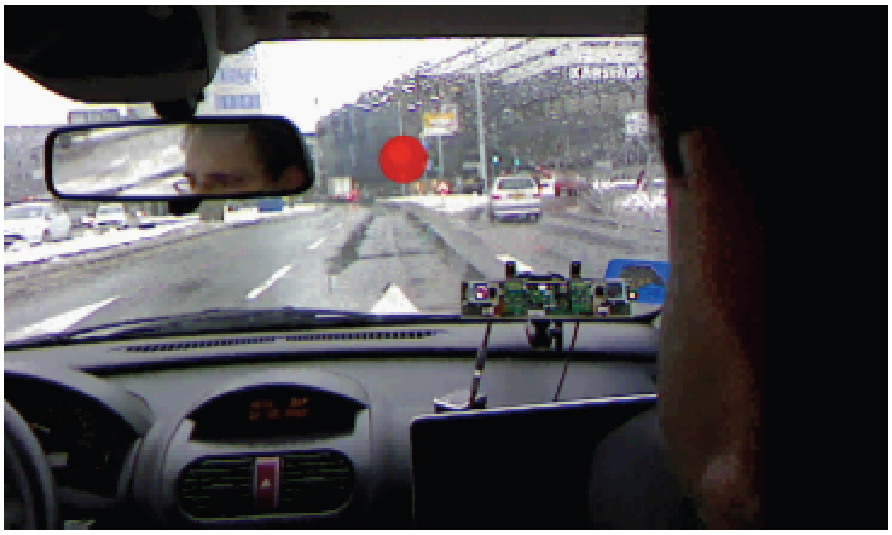


Figure 3.4 – Head and eye-tracking are used for identifying the building in the focus of attention of the co-driver (Moniri and Müller, 2012).

second uses a dwell time approach where the user has to look at an item for a predefined period of time (about 150-250ms). In an experimental application, gazes were used for the post-correction of unconstrained dictation for, e.g., email, twitter, or text messages. For this, a misunderstood word was selected by an eye-gaze and replaced with a word from a list of alternative recognition results.

Toyama et al. (2014) combine a head mounted eye-tracker with an augmented reality system on a head mounted display. The eyes are used to indicate regions of interest in text documents and to activate text recognition and translation functions. The mixed-reality system translates text snippets from Japanese to English and displays the result close to the Japanese text on the head mounted display. Two gaze gestures are proposed for activating OCR text reading and translation. The first strategy is to look at the beginning and the end of the text line alternately and repeatedly (gaze repetitive leap). The second strategy moves the gaze from the beginning to the end gradually (gaze scan).

In Qvarfordt (2005), eye-gaze patterns are used to directly influence the dialogue behaviour on an interaction and not only to apply additional information. The presented iTourist system is an interactive system for city trip planning and exploits findings from a user study of human-human collaboration systems that showed that users' interest can be sensed based on eye-gaze patterns. A city map contains icons for several points of interest, including hotels, restaurants, attractions, nightclubs, bus terminals, and the tourist information office. If a place is presented, the system gives information about the object via speech output and simultaneously shows images of the object. The activation of this presentation is exclusively controlled by eye-gazes. For this purpose, an algorithm was developed that calculates the activation level of every object on the screen.

The level depends on how long and how often a user focuses on an object. Thus, the presentation of an object starts if the activation level exceeds a defined threshold value, respectively it stops if the activation level drops below this threshold again. A second eye-gaze pattern detects whether the user switches back and forth between two places on the map. In this case information about the distance between these two objects is given.

Nowadays eye-gaze control systems are often used for enabling people with disabilities to communicate and interact with the world. They can use their eyes to write texts by looking on the keys of a virtual keyboard that is afterwards synthesised to spoken language. Other systems let them control graphical user interfaces by clicking on buttons or selecting elements on the screen. First implementations of eye-gaze control have already found their way onto the mass market. For example Samsung introduced with their smartphone Galaxy S4 the "smart pause" functionality. This feature recognises with the front camera if the user looks away from the handset and when watching a movie, the device will pause the film.

In the previously presented examples, eye-tracking plays an active role when commands are given to the computer. Stiefelhagen and Yang (1997) propose that eye-gaze can also be passively applied in a multimodal dialogue system. Their presented dialogue system is a system that allows one to work on diverse tasks that are each running in an individual window on the screen. The primary interaction is performed by speech commands. Although today in modern speech recognition systems the problem is not critical anymore, 20 years ago it was important to limit grammar size in order to maintain performance and reliability. To achieve this, a camera-based gaze-tracking system detected the window that was currently in focus of the user. Thus, the grammar model could be reduced to grammar rules that only concern the tasks that are available for this window.

Marshall (2007) describes an approach for identifying the individual's cognitive state from eye metrics and presents a set of metrics that are useful in measuring the cognitive awareness of the user. First is the index of cognitive activity that is determined from raw pupil measurements. This approach observes the high frequency details of pupil changes that are independent from pupillary responses to changes in light which is a slow dilation. A correlation between an increasing index and demanding cognitive processing has been validated across a number of complex cognitive tasks. Other metrics are blinking, the movement of the eyes, and a difference between horizontal location for left and right eye that permits inferences whether the eyes are focused on a specific feature or not. Although this information is not used to directly contribute to human-computer interaction, it provides valuable information about the user's state. In a situation adaptive dialogue system it can be used to adapt the dialogue strategy currently being pursued by the system to the user's attention.

3.1.3 Hand and body gestures

Since the early stages of human development, gestures and body movements have been important channels for communication. In human-computer communication they are also a promising approach for the realisation of natural and intuitive interaction strategies. Starting from the *"Put That There"* system that already supported pointing gestures in a multimodal context, many further research projects have worked on this topic.

Especially hand movements play an important role and can contribute a rich set of information to a conversation. Karam (2009) compared literature about gestural interaction regarding the body parts that have been employed for gesturing and found out that especially hand gestures have been widely used (compare Figure 3.5). Aigner et al. (2012) describe a classification scheme for gesture types. *Pointing* gestures are used to indicate objects, persons or directions. *Semaphoric* gestures are hand postures and movements that have a specific meaning which is mostly dependent on the actor's cultural background and experience. For example, a thumbs-up gesture means "OK" and a flat palm facing hand "STOP". *Pantomimic* gestures imitate a specific action. Often they involve imaginary objects that are not actually present. With *iconic* gestures, information about entities or objects is communicated, e.g., size, shape or paths of motion.

Although gesture interfaces have not achieved a breakthrough in commercial systems yet, with the success in video games like the Wii Remote or Microsoft Kinect and the development of new inexpensive gesture-based input devices like the Leap Motion device, over the last few years principles of gestural interaction have become more and more topical for other domains like the smart home, car or robot control. Wachs et al. (2011) identify three main advantages of gesture applications over conventional human-machine

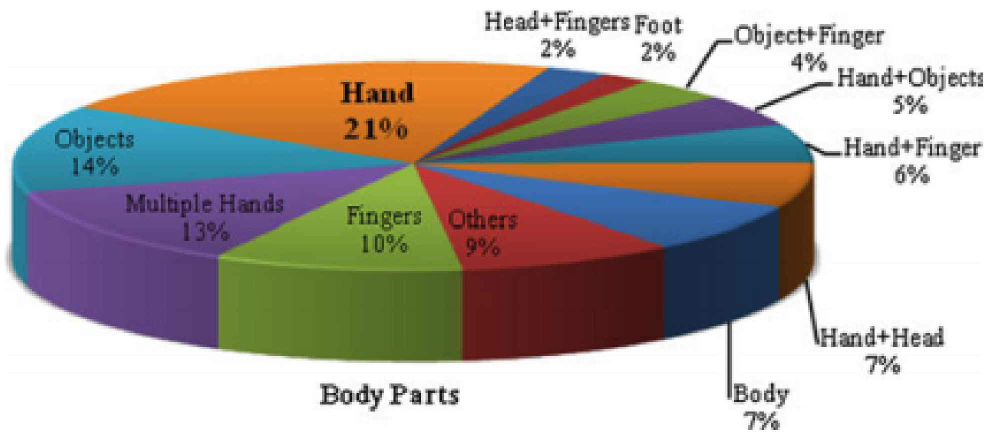


Figure 3.5 – Comparison of body parts that have been employed in literature about gestural interaction (from Rautaray and Agrawal (2015))

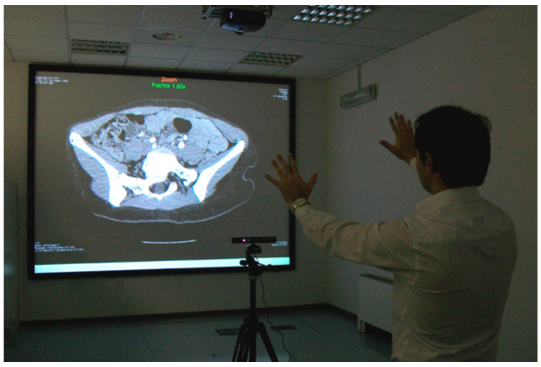
interaction: (i) Touchless interfaces that allow total sterility especially in health-care environments; (ii) Overcoming of physical handicaps for the elderly or people with impaired mobility; (iii) Exploration and manipulation of big data through intuitive actions that benefit from 3D interaction. The following part provides a brief overview of related work (sorted by the application domain) that use gesture based recognition systems for interaction.

Medicine

Gestures have the great advantage that they are touchless. Especially in an operating room where the sterility of the surgeons' hands and the surgical instruments has the highest priority, this provides a better protection against contamination. Graetzel et al. (2004) present the non-contact mouse, a vision-based gesture recognition system that allows the surgeon to navigate through and manipulate images and data visualisations by simulating standard mouse functions with hand gestures. More advanced systems (Gallo et al., 2011; Jacob et al., 2013) support a gesture lexicon of semaphoric gestures which cover a set of functions that have been identified as being useful in an operating room while navigating through radiological images, e.g., by browsing, zooming, rotating and changing the brightness (Figure 3.6a). Both systems use the Microsoft Kinect device for gesture recognition. Wachs (2010) presents an intelligent and collaborative operating room that supports multimodal input and uses computer vision techniques in order to determine the focus of attention of the surgeon by evaluating the surgeon's head orientation, torso posture and orientation. The focus of attention serves two purposes: First, the system can use the gaze orientation for the projection of medical imagery; a steerable projector allows one to use the wall in focus as a display surface. Second, the system can decide whether the surgeon intends to communicate with the patient, the system or another person in the room. The doctor can use two different input modalities for the interaction with the system. A gesture interface allows him to browse medical databases and manipulate the projected radiological images. Speech input extends the input options with functions that are difficult to describe with hand gestures because no natural association exists, e.g., the surgeon says the name of the patient in order to retrieve medical images of him.

Automotive user interfaces

Besides their primary task to control the car, drivers interact with a variety of controls and applications while operating a vehicle. They make use of information, communication and entertainment systems in the car. A great challenge for car manufacturers is to design user interfaces that cover the large variety of functions and to provide interaction concepts that are intuitive and straightforward to use without distracting the driver and endangering his safety. Recent developments focus on gesture interaction with the aim of replacing the classical tangible controls like buttons or levers. Riener (2012) presents


(a) Image Navigation in medicine (source Gallo et al. (2011)

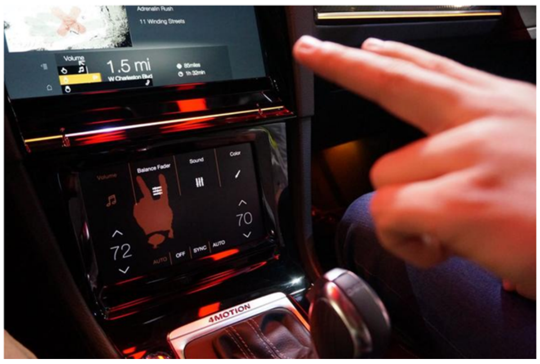

(b) VW Golf R Touch (source www.cnet.com

Figure 3.6 – Example applications for gesture based interaction.

two possible setups for gestural interaction in the car. The first is a capacitive system that tracks finger movements with the hand on the shift knob. The second is vision based and uses a Kinect device mounted on the ceiling for the recognition of static and dynamic fingers as well as hand gestures in the vehicle's gearshift area. Mahr et al. (2011) evaluated finger gestures with the hands on the steering wheel in terms of preference and perceived physical demand. The industry is also starting to embrace gesture interaction in the car. At the CES 2015 in Las Vegas, Volkswagen presented the Golf R Touch, a new touchless gesture-controlled interface where drivers can navigate and activate areas of the dashboard interface by holding their open hand, palm down, over the shift knob (Figure 3.6b).

Human-robot interaction

Robots are machines that look like humans or can perform human functionalities. This fact motivated researchers to apply gestures and body poses as an intuitive way to communicate with robots. Wachs et al. (2011) mention the following advantages of gestures in human-robot interaction: (i) Gestures can be combined with speech and thus improve robustness. (ii) Hand actions involve valuable geometric properties. This is especially helpful for navigational robot tasks, e.g., a pointing gesture with a "move there" command. Furthermore, hand gestures can simulate actions of a robot gripper. (iii) Gesture interaction is very intuitive and brings operability to beginners who are not used to sophisticated robot control commands.

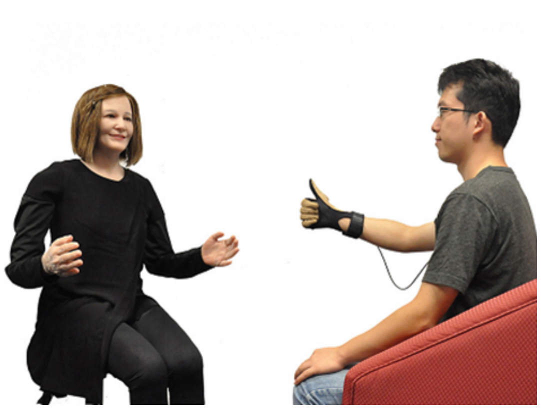
Obaid et al. (2014) describe a system that uses gestures of either one hand, two hands or the complete body to control a humanoid robot. The basis of their work is the Full Body Interaction framework (FUBI), which allows full body user tracking on the data of depth sensors. In a study with technical and non-technical user-groups, they defined an intuitive gesture set for navigational commands that is used to control a humanoid Nao robot. The gesture set includes *move forward, move backward, move right, move left, turn left, turn right, stop movement, speed up, slow down, stand up*, and *sit-down.*

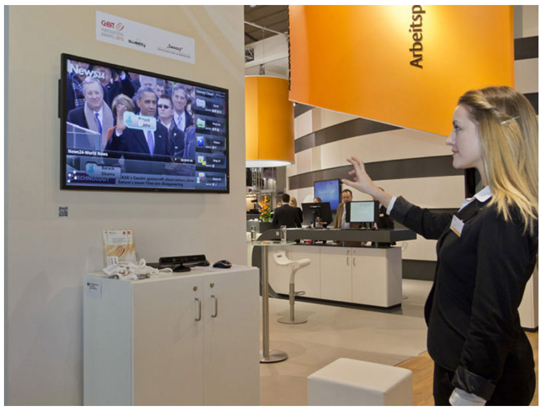
Jokinen and Wilcock (2014) present a framework for multimodal interaction with a Nao robot. In contrast to the previously presented work, gestures are used as an output modality. The main idea is to extend a speech-based conversational system that presents Wikipedia articles with the expressiveness of gestures by experimenting with various gestures that are performed by the Nao robot. One of the main issues here was to synchronise gestures and speech.

Xiao et al. (2014) present the interaction with a humanoid social robot based on body language (Figure 3.7a). In their work the robot understands the meaning of human upper body gestures. For output, it combines body movements with speech and facial expressions. Thus, their approach follows the symmetric multimodality paradigm by Wahlster (2003). Gestures are recognised by the early fusion of data from a CyberGlove II for capturing hand positions and a Kinect for the detection of upper body poses. The controlled robot is a realistic human-sized robot with 27 degrees of freedom for displaying natural looking movements. In their scenario the robot interacts with the user in a classroom. The robot takes the role of a lecturer and the user is the student. The robot is able to understand and interpret 12 different symbolic or pantomimic upper body gestures. For example, a raised hand is interpreted as "I have a question", a pantomimic drinking gesture asks for a permission to drink. The reaction of the robot is a combination of verbal and nonverbal output where the latter consists of bodily emotional expressions that emphasize the spoken answers.

Smart Home and Ambient Assisted Living

In the domestic domain, user interfaces are often distributed throughout the room. Some are simple like light switches, others complicated like remote controls for the TV. Many projects have examined the use of gesture-based interaction in smart homes, e.g., the Swoozy Semantic Television System (Deru and Bergweiler, 2014) can be controlled by hand or finger gestures (Figure 3.7b). Kühnel et al. (2011) access the accelerometers of a smartphone for the recognition of dynamic arm gestures in order to control home functionalities like lamps, TV, video recorder, or the window shutters. In a study they tried to find intuitive gestures that can be used for the control of home appliances. One of their findings was that one gesture should be used to control several devices in order to shrink the gesture vocabulary. The referred device could be selected either by pointing or by pressing a certain area on the mobile device. Neßelrath et al. (2011) go a step further and classify functionalities into categories. All functionalities in one category have a common meaning. For example, the category 'turn on' can be applied to many devices, e.g., a television, a lamp but also the a hood fan. The category 'increase' includes, amongst others, the volume of a television or the fan speed of the extraction hood. The finally triggered functionality is dependent on the current context of the discourse. In the demonstrator system, which uses a Wii Remote Controller instead of


(a) Interaction between a human and a humanoid robot (source Xiao et al. (2014))



(b) Gesture based interaction in the Swoozy system (source *www.swoozy.net*)

Figure 3.7 – Example applications for gesture based interaction.

a smartphone, the context was set by speech input. If the user mentions the TV while performing a 'turn on'-gesture, the TV is activated. If he mentions the lamp, instead, the light is turned on.

Physically impaired people are often incapable of performing natural body gestures. Nevertheless, gestures may improve their possibility to control devices in their environment. The set of gestures in this case must be more personalised to the specific person and regard the parts of their body they can control and the degree of freedom of the movements. Hirsch et al. (2014) for example present a novel sensing modality for hands-free gesture controlled user interfaces. They integrate four capacitive electrodes into a textile neckband and use active capacitive sensing for continuous unobtrusive head movement monitoring. The device is capable of recognising head movements like *nod, tilt, look, circle* and *woodpecker move*. The system can be used for recognising natural communicative behaviour like nodding in order to affirm something. Disabled people who can only move their heads can now operate arbitrary devices if the head gesture vocabulary is appropriately defined.

Krieg-Brückner et al. (2010) present with the Bremen Ambient Assisted Living Lab (BAALL) (see Figure 3.8) a smart environment for people with diminishing physical and cognitive faculties. This includes the Rolland wheelchair, a mobility assistant that can automatically navigate through the environment and automatically avoids any obstacles and assists the user when going through doors. A safety assistant ensures that the vehicle stops before a collision can occur. This is realised using laser range sensors. Furthermore, BAALL contains smart furniture that adapts to the users' needs like a kitchen counter that can be moved up and down or cabinets that can be individually moved. The communication between the user, smart devices, and other assistants is intended to be natural and effective. Due to the complexity of the systems to control and the users' impairments, new multimodal interaction concepts are tested in order to introduce the

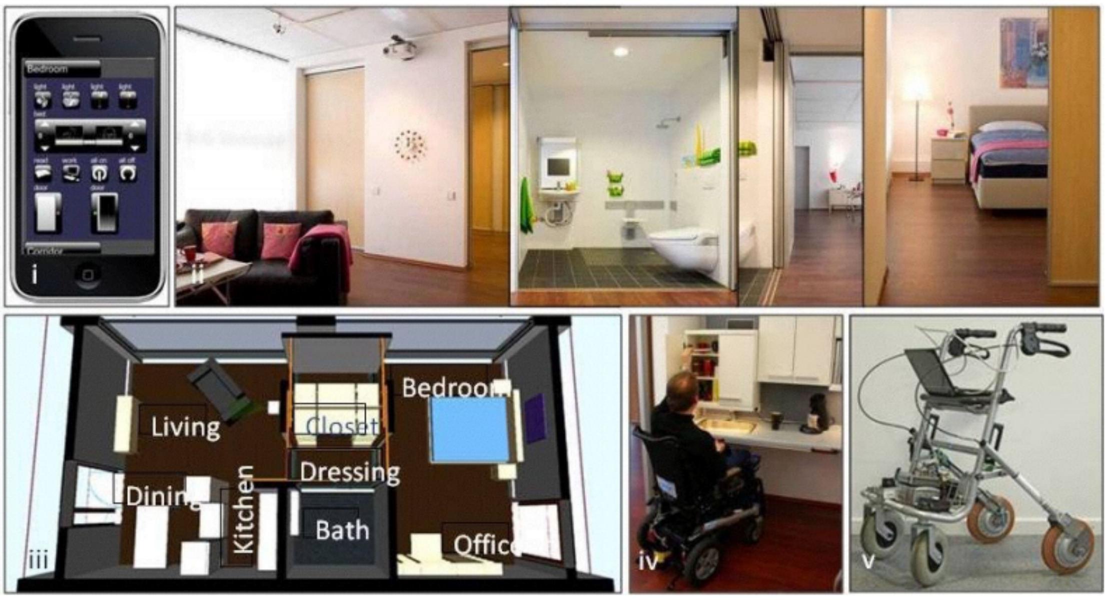
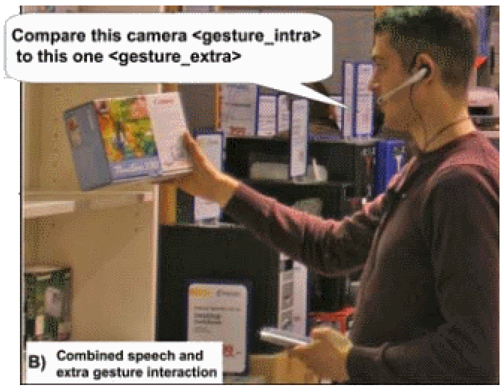


Figure 3.8 – Infrastructure of the *Bremen Ambient Assisted Living Labs* (from Frey et al. (2010b))

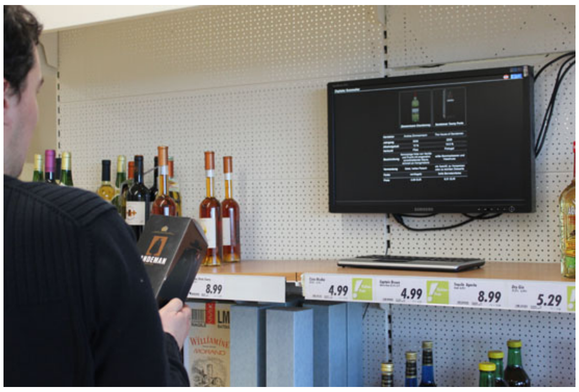
system into people's daily living. This includes natural language dialogues, visualisation, gestures via touch screen, and a head joystick. By the combination of several services it is possible to create higher-level services. One example presented is the utterance *"I'd like to eat a pizza"*. After interpreting the user's intention the smart environment can proactively plan the next actions. While the mobility assistant navigates the user to the kitchen (which includes opening the doors, planning a route, etc.), the smart furniture adapts to the user's need and moves the microwave downward in order to make it accessible to the user who sits in the wheelchair.

Innovative interaction in the retail domain

Especially shopping scenarios provide reasonable use-cases for physical acts with tangible user-interfaces. Wasinger et al. (2005) show how the interaction with real world artifacts can be multimodally combined with input that is based on speech, handwriting, and gestures. They describe a demonstrator called Mobile ShopAssist that supports the customer with additional product and product comparison information while he is shopping in a store. The shelves and products are instrumented with RFID technology and allow the users to directly refer their information request to the products in their hands. For this they define the two concepts *intra-gestures* and *extra-gestures*. The intra-gestures are on-device interactions, in the scenario on the touchscreen of a mobile device. They are used to introduce products that are displayed on the screen into the discourse by pointing gestures. Extra-gestures are off-device interactions and


Compare this camera <gesture_intra> to this one <gesture_extra>
B) Combined speech and extra gesture interaction

(a) The Mobile Shop Assistant (from Wasinger et al. (2005))



(b) The Digital Sommelier (source *http://www.innovative-retail.de*)

Figure 3.9 – Physical acts with tangible user-interfaces in the retail scenario

occur when the user interacts with artifacts in the real world. In the demonstrator, the RFID technology is used in order to detect "pick-up" and "put back" actions. The user can combine speech with intra or extra-gestures and thus retrieve information about products either by referring to them via pointing gesture on the touch-screen or by picking them up from the shelf. Furthermore, it is possible to compare real world artifacts with objects presented on the screen by combining intra and extra-gesture within one multimodal interaction (see Figure 3.9a).

Schmitz et al. (2008) present an interactive wine shopping assistant with the name *Digital Sommelier*. The shop assistant provides an intuitive multimodal interface for retrieving general product information but also particular attributes of a specific product, such as the temperature. The wine bottles are instrumented with wireless sensors, e.g., for retrieving the current temperature. Furthermore, RFID and acceleration sensors are used in order to detect physical user interactions with the bottle. Combined with natural language, the customer can retrieve information about a wine bottle he picked up off the shelf. If the user takes a bottle off of the shelf, the Digital Sommelier generates a corresponding product information page and displays it on a nearby screen. Simultaneously the product is introduced using speech synthesis technology. The application also receives the current temperature of the wine, compares the actual temperature against the proposed temperature at which the wine should be served, and generates appropriate tips. If the user turns the wine in order to read the information on the rear side of the bottle, the rotation is detected by the accelerometer sensors and the Digital Sommelier changes the information displayed on the nearby monitor. In this case the web page of the manufacturer is referred, which provides additional information to the customer.

3.2 Multimodal Dialogue Frameworks

The previous section introduced some applications in several domains that benefit from multimodal interaction. For the development of new applications, it is of advantage to use frameworks that support the development process in technical terms as well as in the provided design patterns and strategies that support the multimodal integration. In this section some such frameworks are examined.

3.2.1 AT&T speech mashup architecture

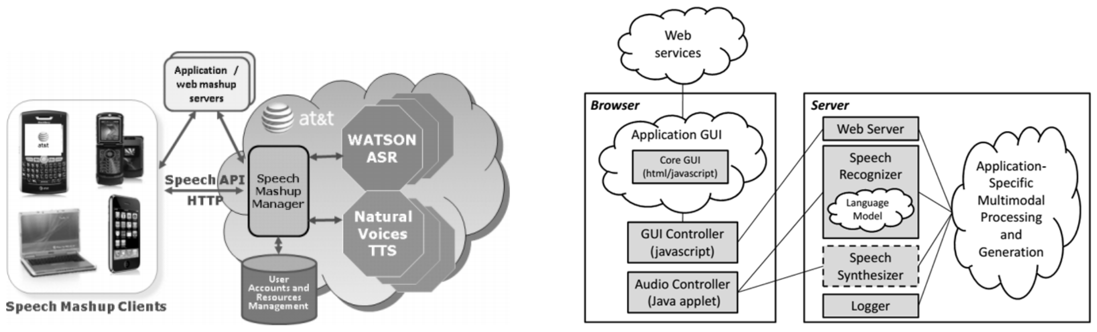
The AT&T speech mashup architecture (Di Fabbrizio et al., 2009) aims to simplify the combination of web content with speech processing and makes speech recognition and TTS synthesis available by web services. The main idea is the support of speech practitioners and researchers in the easy and rapid development of speech and multimodal mobile services. The architecture consists of four components:

Speech Mashup Server (SMS) - This server provides the services for speech recognition and speech synthesis. Furthermore, the server contains the AT&T speech mashup server that handles the direct connections between the connected client devices, including resolving device-dependency issues, performing authentication, and general accounting.

Speech Mashup Client - A client application that runs on the mobile device which presents the UI (e.g., iPhone, Safari browser). The client sends and receives audio buffers for speech input and output and receives the recognition results from the server.

Main Application Server - A web service (e.g., Apache or Tomcat). Depending on the application, the server provides access to a back-end database, collects and aggregates data from other application servers and could implement the application logic.

A mobile client application establishes connections to the application server and the SMS. Between the client and SMS, data and speech are exchanged. The speech either consists of the voice of the user that is recorded for speech recognition and sent to the server, or the synthesised speech output that is sent to the client for audio output. The speech processing resources are centrally managed in a web-based portal that supports rule-based and stochastic grammars. For rule-based grammar specification, the Speech Recognition Grammar Specification (SRGS) standard is used. One task of the web portal is the grammar management which allows one to extend the shared grammar rule-set with personalised user grammars. For the representation of recognition results, three different language formats are supported: XML, JavaScript Object Notation (JSON),


Application / web mashup servers
at&t
WATSON ASR
Speech API
HTTP
Speech Mashup Manager
Natural Voices TTS
User Accounts and Resources Management
Speech Mashup Clients
Web services
Browser
Application GUI
Core GUI (html/javascript)
GUI Controller (javascript)
Audio Controller (Java applet)
Server
Web Server
Speech Recognizer
Language Model
Speech Synthesizer
Logger
Application-Specific Multimodal Processing and Generation

(a) Speech Mashup Framework (from Di Fabbrizio et al. (2009))

(b) WAMI Toolkit (from Gruenstein et al. (2008))

Figure 3.10 – Architecture comparison of the the WAMI toolkit and the speech mashup framework. In both architectures a client connects to server components for speech recognition, speech synthesis, and application processing. The WAMI toolkit additionally manages the GUI presentations.

and the EMMA markup language, a W3C standard for the interoperable input representation in multimodal systems (see Section 3.3.2). TTS tasks are described with the W3C Speech Synthesis Markup Language (SSML).

The applications that have been implemented with this framework range from speech based information retrieval from a business directory to the ordering of food from a pizza service. The applications combine graphical interfaces with speech interaction, the possible input is a mix of voice and touch inputs. Nevertheless, the support of cross-modal interaction is not mentioned.

3.2.2 WAMI toolkit

The Web-Accessible Multimodal Interfaces (WAMI) toolkit (Gruenstein et al., 2008) provides a framework for the web-based development, deployment, and evaluation of multimodal user interfaces. The supported interaction channels are speech input, speech output and graphical user interfaces that can be controlled by mouse, pen or touch, depending on the used device. The main goal of the framework is to be able to rapidly build applications that are accessible from outside a laboratory in order to more easily perform user evaluations. A secondary goal is that the toolkit supports a lightweight development model for non-expert dialogue application developers in order to build interactive multimodal applications. The framework supports desktop, laptop, and tablet PCs.

The framework uses a server-client architecture where the main application runs on the server. On the client, a web browser presents the GUI application in HTML and AJAX while an audio controller handles the audio stream between device and server. Speech recognition and synthesis are realised on the server. Thus, the client only has the task to establish the interface to the user; the program logic is located on the server. Grammars are specified using the Java Speech Grammar Format (JSGF), GUI information is either sent in final HTML that is rendered in the browser or in XML messages that update the content of an application specific GUI on the client. As example applications, some web-based showcases are presented that support speech based interaction as well as interaction with the GUI. Cross-modal interactions are not included.

3.2.3 DIANE

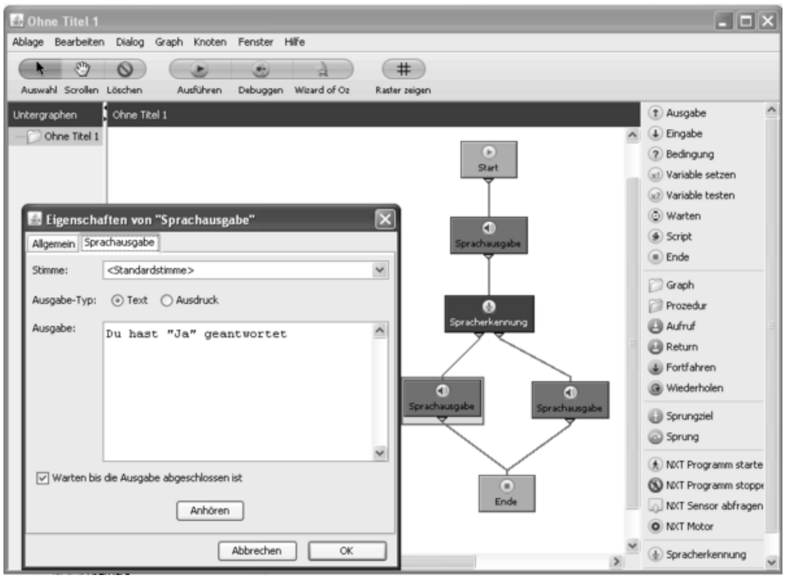
The speech dialogue system DIANE (DIAlogmaschiNE) is a frame-based dialogue system for speech interaction developed by Siemens CT (Song, 2006; Block et al., 2004). Applications are modelled with DIANE as a set of transactions, where a transaction is considered similar to a frame. Each function provided by the backend application corresponds to one transaction in the dialogue model. Additional information that is required for the execution of a transaction is modelled as parameters of the transaction. At the example of a transaction for train ticket reservation, these parameters are information about departure, destination, date, and time of the itinerary. For each of these parameters an own grammar and prompts for query or confirmation are defined. Furthermore, trigger grammars for the direct access to the transactions can be defined.

Siemens developed DIANEXML, an XML-based dialogue design language for supporting the development process of speech user interfaces. This design language is used to automatically generate the runtime resources for the DIANE dialogue system. A dialogue application is specified by three kinds of documents. With a transaction file the application is defined as a set of executable transactions. Additional necessary information is defined in a parameter file the transaction file can refer to. The grammar files provide important information for speech recognition and language understanding. Further functionalities like callback functions in order to specify inference rules, consistency conditions, repair mechanisms and to invoke backend functions, are implemented by a Java interface.

The framework focuses on the frame-based dialogue management approach, which allows one to build mixed-initiative dialogue applications. The support of the resolution of discourse phenomena and integration of additional modalities are not mentioned.

3.2.4 Dialog OS

The Dialog OS (Bobbert and Wolska, 2007) is an extensible platform for the development of dialogue systems with the focus on spoken language. It is an educational tool that allows students with even non-technological background to develop relatively complex applications with flexible strategies for various domains. It is a commercial product that has been developed by the former CLT Sprachtechnologie GmbH.



Eigenschaften von "Sprachausgabe"

Stimme: <Standardstimme>

Ausgabe-Typ: [✓ Text] Ausdruck

Ausgabe: Du hast "Ja" geantwortet

☑ Warten bis die Ausgabe abgeschlossen ist

Figure 3.11 – The dialogue specification workbench in DialogOS (source: http://www.debacher.de/wiki/DialogOS)

The system mainly focuses on speech based dialogues but can be extended with new components over a communication API, for example actuators like a LEGO Mindstorm robot or an elevator. The dialogue modelling approach is based on a Finite state automaton (FSA) that can be built in a GUI workspace (see Figure 3.11) where dialogue nodes of the FSA are created and linked to each other. Nodes can be input nodes, output nodes or nodes that perform internal actions. The latter can be the execution of a JavaScripts or the assignment of variables. Furthermore, sub-automata can be executed that describe recurring parts of the dialogue. Input nodes allow one to define a list of expected input values either as plain text or regular expression. If an incoming text- or speech-based input message matches the value, the FSA follows the outgoing edge to the next node. Output nodes can be used to trigger TTS tasks.

3.2.5 SmartKom

The SmartKom project (Wahlster, 2003, 2006b) was one of the largest projects worldwide that examined multimodal interaction. It was funded by the German Federal Ministry of Education and Research (BMBF) and consisted of over 10 consortium partners led by the DFKI. The result of SmartKom was a multimodal dialogue system that combined speech, gesture, and facial expression for input and output (Figure 3.12). One focus was set on symmetric multimodality. For this, a virtual character was used to communicate with the user via speech, pointing-gestures, and facial expressions by imitating human characteristics. For input, the system supported gesture input recognised by an infrared camera, speech recognition, and a camera for facial analyses. Handwriting and hand contour recognition were used only during biometric signature identification. Furthermore, physical actions were recognised and used in order to validate whether the user executed a correct action in order to complete a collaborative task. SmartKom was built on an architecture for distributed components called Multiplatform which is an open, flexible and scalable software architecture that allows one to integrate heterogeneous software modules written in diverse programming languages and running on various platforms. In total, 40 asynchronously running modules were included in SmartKom. The data interface between the components is covered by the M3L (Multimodal Markup Language) which is described in detail in subsection 3.3.1.

The integration and mutual disambiguation of multimodal input and output is solved by statistical and symbolic methods that are processed on both the semantic and pragmatic level. The uncertainty and ambiguity during the analysis of the various input modalities is corrected with the help of scored hypothesis graphs that stem from the user interactions. The speech recogniser provides word hypothesis graphs, the prosody component clause and sentence boundary hypothesis graphs, the gesture recogniser hypotheses about possible referenced objects, and the facial expression interpreter about the emotional state of the user. They are unified by the fusion component using unification and overlay and the resulting interaction hypotheses ranked by the intention recogniser. Here also, the particular context of the multimodal discourse model is considered making the final ranking highly context sensitive. The presentation planning in SmartKom is solved with a plan-based approach. The presentation goal for the planner is encoded in a modality-free representation. The goal is recursively decomposed into primitive presentation tasks. For this, the system contains 121 presentations strategies that are parametrised by the discourse context, the user model, and the environmental context. Finally, each presentation task is sent to the appropriate generator for concrete output realisations.

The reference resolution in SmartKom is based on a three-tiered multimodal discourse model. This consists of a discourse layer, a domain layer, and a modality layer. The discourse layer stores information about every discourse object mentioned. Since it is multimodal, this comprises verbal as well as visual and the conceptual context which includes all visible objects on the screen and the spatial relationships between them. Each of the stored objects can have multiple surface realisations on the modality layer. Each element in the discourse layer is also linked to an instance in the ontology-based domain model of SmartKom. The discourse model allows one to address the following multimodal dialogue discourse phenomena: *multimodal deixis resolution and generation, multimodal anaphora resolution and generation, cross-modal reference-resolution and generation, multimodal ellipsis resolution and generation.* Furthermore, the dialogue management supports strategies for *turn-taking* and *back-channeling.*

One goal of the SmartKom project was to integrate the previously presented multimodal capabilities in a dialogue shell that is reusable, efficient, and robust. With a flexible configuration, domain independence and a plug-and-play functionality, it was possible

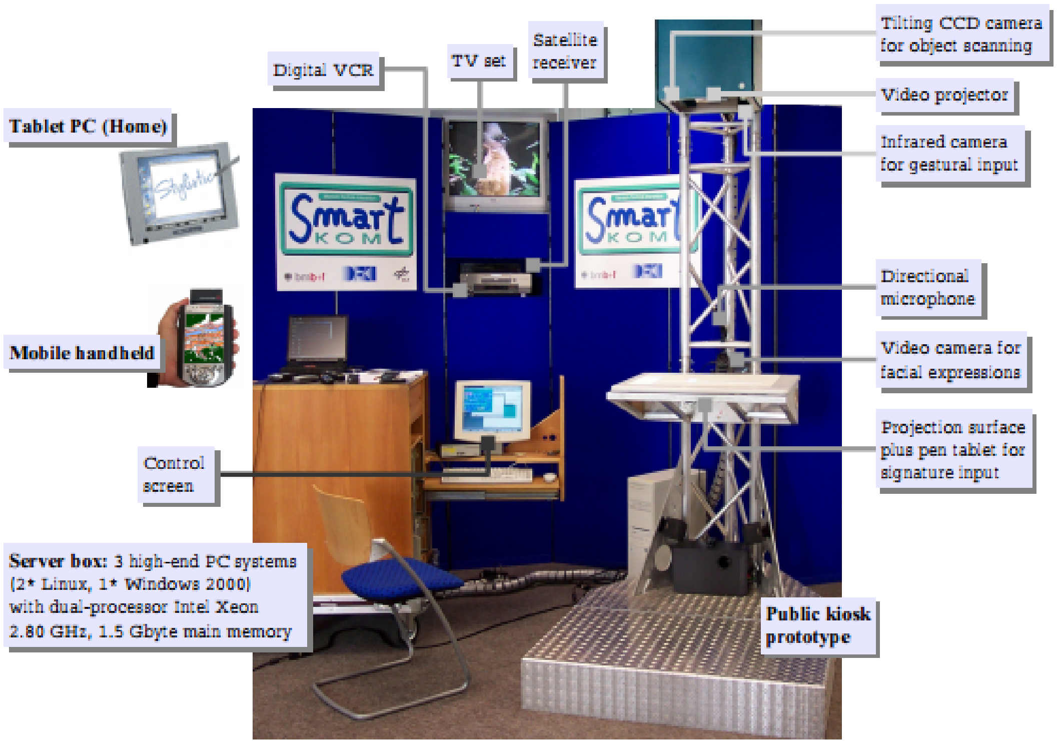

Digital VCR
TV set
Satellite receiver
Tilting CCD camera for object scanning
Video projector
Infrared camera for gestural input
Tablet PC (Home)
Directional microphone
Mobile handheld
Video camera for facial expressions
Projection surface plus pen tablet for signature input
Control screen
Server box: 3 high-end PC systems (2* Linux, 1* Windows 2000) with dual-processor Intel Xeon 2.80 GHz, 1.5 Gbyte main memory
Public kiosk prototype

Figure 3.12 – The SmartKom reference installation for demonstrations (c.f. Wahlster (2006b))

to simply adapt the system to new scenarios. The three scenarios that were implemented under the SmartKom project are the following (Wasinger, 2006):

SmartKom-Public - A multimodal public communication kiosk application for domains like train stations and airports. Input can be provided by speech, facial expressions, and gestures on a touch screen. Output is provided by a virtual character that combines multimodal interaction in the form of graphical output, speech, and gestures.

SmartKom-Mobile - A mobile travel application on a PDA for navigation and point-of-interest information retrieval. The interaction is adapted to the limited resources of the PDA. Input is provided by speech and pen-based pointing, output by a simplified version of the virtual character agent.

SmartKom-Home/Office - The user can use multimodal interaction in the form of speech combined with gestures in order to control an infotainment system. The scenario includes information retrieval from an electronic program guide and the control of consumer electronic devices like TVs, VCRs and DVD players.

3.2.6 ODP

The Ontology-based Dialog Platform (ODP) framework (Porta et al., 2014a; Neßelrath and Porta, 2011) evolved from the SmartKom dialogue shell and was developed with the aim to enable developers to easily design and implement homogeneous ODP-based UIs for services. The complexity of the back-end architecture should be hidden from the user. The development mainly took place during the THESEUS research program that was funded by the Federal Ministry of Economics and Technology with the goal of developing new technologies and methodologies for the Internet of Services.

The ODP framework provided a platform and development methods for the rapid creation of multimodal dialogue applications with the main task to retrieve heterogeneous data from web services (Sonntag et al., 2009). A declarative programming approach helped UI experts, who are not necessarily Java programming experts, to build applications. A main concern was to offer more abstract and comprehensible work levels instead of working on the most concrete level of abstraction. For example, the platform provides an advanced domain-independent grammar specification language. The language allows one to directly map utterances to a semantic interpretation of the input. Furthermore, algorithms are supported that handle the dynamic generation and management of named entity grammars. Rule-based speech synthesis templates allow a generic output generation strategy. A GUI model that can directly be connected to the semantic data model fulfills the paradigm "no presentation without representation" and is used to represent interaction-relevant graphical components. The platform out of the box supports the multimodal integration of speech recognition, speech synthesis and interaction with GUIs. Therefore, concepts for dialogue management, modality fusion, and discourse resolution have been adopted from the predecessor project SmartKom.

All content in ODP is semantically modelled. This is a necessary requirement for the robust multimodal processing and dialogue management. Information that is retrieved from integrated services must first be transformed into a semantic representation and mediated back into the internal domain model that is based on extended Typed Feature Structures (eTFSs) (see also Section 4.2). Thus, several multimodal dialogue UIs have been implemented for scenarios in various application domains:

TEXO (Porta et al., 2009a) - A mobile business application for the management of business processes. A decision-maker can handle purchase order requisitions in an enterprise resource planning system. Alternative products are visualised in three-dimensional space. The interaction is achieved by speech and GUI on an iPhone.

MEDICO (Sonntag and Möller, 2009) - A stationary medical system for the retrieval of diagnoses, semantic annotation of radiological images, and comparison of patients' findings.

CALISTO (Bergweiler et al., 2010) - A collaborative kiosk infotainment system that combines mobile interaction with the interaction on a tabletop surface. The

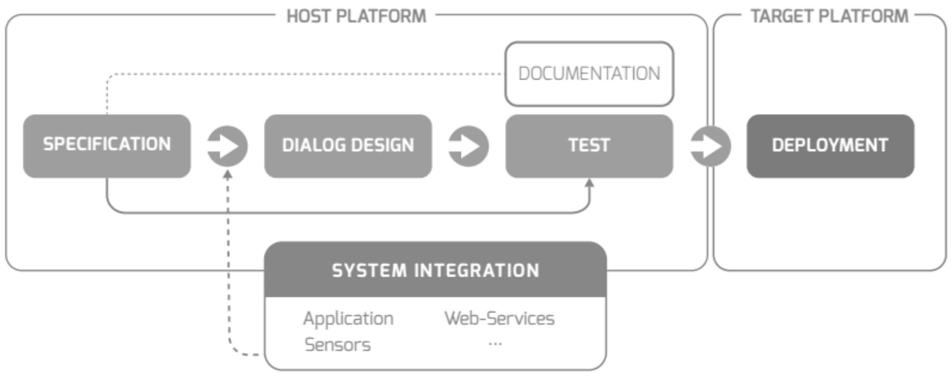


Figure 3.13 – The tool chain of the ODP S3 workbench (c.f. SemVox (2015))

scenario comprises information from several application domains that is retrieved by accessing a large heterogeneous service back-end including semantic web services.

Since the end of the project, ODP has been further developed by the DFKI spin-off company SemVox GmbH[1] and is sold under the name ODP S3. Heavy improvements have been made regarding the software development kit with the aim that "developing dialog components becomes as easy as developing apps for a smartphone" (SemVox, 2015). The extended ODP S3 workbench provides an integrated tool chain (see Figure 3.13) based on Eclipse and supports all steps of the development process, from the specification of dialogues, over system integration, to an improved quality management for commercial and professional use. This, e.g., includes automatically generated test cases and system documentation. The platform itself demands low resources and is available for the target platforms Linux x86/ARM, QNX Neutrino, Android, Windows Embedded, and more. The supported features have also been extended with goal-oriented interaction, task-based dialogue models, hybrid speech recognition and the consideration of personalisation, user models, and cognitive load.

3.2.7 Cue-me

The CueMe^{TM} (Openstream, 2015, 2011) software development platform is developed and licensed by Openstream and allows one to create multimodal systems based on the W3C international standard reference architecture (Dahl, 2013). This standard embraces EMMA, a standard for content in messages for multimodal interaction (see Section 3.3.2). The framework is based on the OSGi standard.

The framework collects user input from diverse modalities like type, touch, talk, gestures, handwriting, or stylus and can also incorporate sensor information from on-device peripherals like camera, GPS, card reader, and bar-code scans. The information can

[1]www.semvox.de

be distributed across multiple devices, like smartphones, tablets and PCs whereby all current operating systems are supported.

The system is context-aware and allows one to consider information from “hard sensors”, such as location awareness, and “soft sensors” such as user preferences. For example, if the user is moving (as in a car) the framework allows one to use speech interaction technology for navigation and control. Multimodality increases the robustness of applications by reducing data entry errors and common mistakes. One modality of input is hereby used to validate the content of other input modalities, which is called “mutual disambiguation”.

An interaction manager is responsible for the coordination of information. For this, the “Openstream MAM Server” holds application definitions that help to map collected input information to appropriate methods on connected application servers. A wide variety of enterprise SOA (Service Oriented Architecture) backend systems are supported.

The spectrum of applications covers form filling, question answering, personal information management, annotation of drawings and more and is settled in domains like health-care, financial services, media and entertainment, and utility and transportation sectors (Oviatt and Cohen, 2015a). Various corporations like Walmart, Merck, Roche, and the Bank of NY/Mellon are mentioned to have implemented multimodal applications based on the CueMe framework.

3.2.8 Summary and Conclusion

In this subsection, several multimodal dialogue frameworks are presented. The individual frameworks focus on specific aspects, Table 3.1 gives an overview. It is intended that the multimodal dialogue platform presented in this thesis incorporates the identified aspects in a comprehensive approach. The following aspects have been identified in detail:

Resolution of Dialogue Discourse Phenomena - The framework provides models and concepts that support the resolution of dialogue discourse phenomena like anaphoras, ellipsis or deixis.

Cross-modal Reference Resolution - A fusion component combines the content of two modalities for cross-modal interaction.

Cross-modal Fission and Presentation Planning - The system supports strategies for presentation planning and multimodal output presentation.

Distributed Devices - Input and output can be distributed over several devices.

Physical Acts - Physical acts are intended to be part of HCI.

Dialogue Specification - The framework supports a strategy for the specification of dialogues.

Language Standards - The framework includes international language standards if available.

Centralized Grammar Management - The grammar resources should be centrally managed, context adaptive, and distributed to connected speech recognition engines.

Extensibility - The framework is easily extendable with new devices and modalities.

Reusability and Domain Independence - Strategies, concepts and components are easily and rapidly adaptable to new applications and domains.

Development Platform - The framework provides a comprehensive development platform that supports even non-expert developers in the creation of multimodal dialogue applications.

	Discourse Phenomena	Cross-modal References	Fission	Distributed Devices	Physical Acts	Dialogue Specification	Language Standards	Centralised Grammar Management	Extensibility	Reusability	Development Platform
AT&T SMA							x	x		x	
WAMI			x					x		x	
DIANE						x		x	?	x	?
Dialog OS						x				x	x
SmartKom	x	x	x	x	x				x	x	
ODP	x	x	x			x	x	x	x	x	x
CueMe		?	x	x			x		x	x	x
SiAM-dp	x	x	x	x	x	x	x	x	x	x	x

Table 3.1 – Comparison of the multimodal dialogue frameworks. Each framework addresses a specific range of important aspects. For entries with a '?' the literature does not provide enough information for the classification.

Although some of the frameworks emphasize the extensibility with new modalities and devices, the main focus in the presented example applications was mostly on the combination of speech and graphical user interfaces. The strengths of the AT&T SMA, WAMI toolkit, and DIANE are modular and flexible architectures especially for speech based dialogue applications, where WAMI and AT&T also integrate interaction with a GUI. The disadvantages are the missing multimodality with more than two modalities and the support of cross-modal reference and discourse phenomena resolution, as well as the fission of output presentations. Systems that contain these features are the SmartKom and the ODP framework. While SmartKom already distributes interaction over several devices and includes physical acts, the ODP system is mostly used for the combination of speech with GUIs.

In contrast to the SmartKom system, ODP and the CueMe system have included a development platform. Furthermore, ODP integrates concepts for the specification of dialogues that support the developer in application development. Similar concepts are introduced by the DIANE and the Dialog OS systems. While most of the presented frameworks use proprietary languages, AT&T SMA, ODP, and CueMe use language standards, e.g., for the description of grammar specifications, speech synthesis and multimodal interaction.

The work presented in this thesis adopts the strengths of the diverse related work frameworks and fills their gaps. Thus, we developed an extensible dialogue platform for the creation of dialogue applications and provide strategies for the specification of dialogues. The first three related works showed that a centralised grammar management component is useful for a context and user dependent grammar recognition. In the presented SiAM framework, we also follow this approach. The representation of grammars, speech output and multimodal interaction are based on international standards. In order to also support more sophisticated dialogues the platform supports cross-modal references and dialogue phenomena. Since the focus of the work is set on Cyber-physical Environments (CPEs), a support of massively multimodality, distributed devices, and the support of physical acts is heavily supported. A further advantage of SiAM-dp in contrast to commercial frameworks is its openness.

3.3 Representing Multimodal Interaction

Many components are typically involved in multimodal dialogue processing, including components for connecting input and output devices, dialogue management, multimodal integration and fusion, and presentation planning. Unfortunately, all these distinct components often use heterogeneous, sometimes proprietary, protocols for information transfer, making the communication between them highly complex and expensive due to the high overhead of model transformations. In this section several approaches are presented that aim to overcome this problem by developing meta languages for the representation of multimodal input and output.

3.3.1 M3L - Multimodal Markup Language

M3L (**M**ulti**m**odal **M**arkup **L**anguage) (Wahlster, 2003, 2006a) has been developed for the SmartKom project (see Section 3.2.5). It is based on XML and comprises all data interfaces within a complex multimodal dialogue system. Therefore, all sub-structures of the particular pools involved in the system are coherently integrated into the language specification. For this, the definition of the M3L language is decomposed into 40 schema specifications. The idea is that during the particular processing steps from user input to system output, the information of a message is continuously enriched with additional content.

The input representation of M3L allows one to provide information about input segmentation, the confidence of recognition and interpretation results, and interpretations of the user's input intention that contain discourse relevant parts as well as knowledge instances or references. On the output-side, M3L supports the modality-independent representation of the system's communicative intention. A media fission component and unimodal generators and renderers extend this representation with concrete descriptions of the corresponding linguistic and visual objects for the surface realisation.

Semantic knowledge representation in M3L is supported with the ontology language OIL (Fensel et al., 2001) that is transformed into a format compatible to M3L. It is represented with XML schemata and can be handled as typed feature structures allowing one to apply unification and overlay algorithms on instances encoded in M3L.

3.3.2 EMMA

The Extensible MultiModal Annotation markup language (EMMA) is an XML-based W3C standard for describing the interpretation of multimodal user input (Johnston, 2009). It is in the state of a W3C Recommendation with the latest version published[2] in February 2009 (Johnston et al., 2009).

The general idea of EMMA is the representation of information that has been automatically extracted from user input by interpreters. Especially the message history of the different stages during input processing from raw-signal, over input interpretation to multimodal integration can be logged.

For the representation of an interpretation, no fix specification is defined. Moreover the standard allows one to integrate application specific markups into therefore assigned containers, giving the required degree of freedom for the content. Further concepts allow one to group mutually exclusive, sequential, or in some manner related inputs. Interpretation instances that are derived from other interpretation instances during the process of the increasing representation refinement from raw data to interpretation can be linked.

[2]http://www.w3.org/TR/emma/

A set of annotation attributes and elements allows one to provide additional meta-data about the input event. This includes information about media-type, signal-format, recognition and interpretation confidence, input source, timestamps, medium, mode and others. For multimodal integration it is possible to define *hooks* which serve as place-holders for content from other input instances. Furthermore, the language allows the extension with non-standardised input annotation with custom vendor or application specific information.

Examples

The EMMA document depicted in Figure 3.14 shows a markup that could have been produced by a speech recognition and natural language understanding component in

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma" ...>
        <emma:one-of id="r1"
                emma:medium="acoustic" emma:mode="voice"
                emma:function="dialog" emma:verbal="true"
                emma:start="1241035886246"
                emma:end="1241035889306"
                emma:source="smm:platform=iPhone-2.2.1-5H1_1"
                emma:signal="smm:file=audio-416120.amr"
                emma:signal-size="4902"
                emma:process="smm:type=asr&version=asr_eng_2.4"
                emma:media-type="audio/amr;rate=8000"
                emma:lang="en-US" emma:grammar-ref="gram1"
                emma:model-ref="model1">
          <emma:interpretation id="int1"
                emma:confidence="0.75"
                emma:tokens="flights_from_boston_to_denver">
                  <flt><orig>Boston</orig>
                  <dest>Denver</dest></flt>
          </emma:interpretation>
          <emma:interpretation id="int2"
                emma:confidence="0.68"
                emma:tokens="flights_from_austin_to_denver">
                  <flt><orig>Austin</orig>
                  <dest>Denver</dest></flt>
          </emma:interpretation>
        </emma:one-of>
        <emma:info>
          <session>E50DAE19-79B5-44BA-892D</session>
        </emma:info>
        <emma:grammar id="gram1" ref="smm:grammar=flights"/>
        <emma:model id="model1" ref="smm:file=flights.xsd"/>
</emma:emma>
```

Figure 3.14 – EMMA sample document (taken from Johnston (2009))

a flight booking system. On the first level of the root element (`emma:emma`) it contains a container `(emma:one-of)` element that holds two alternative interpretations (`emma:interpretation`) for a speech input. One attribute of an interpretation is the `emma:confidence` that reflects the interpreter's confidence with the recognition result. A second attribute is a list of `emma:tokens`, here the term token is used in the computational and syntactic sense of *units of input.* The syntax for these tokens is not predefined, in the example it is the list of words and phrases that have been recognised by the speech recogniser. The element `emma:interpretation` itself is a container for a single interpretation represented in an arbitrary application specific markup, in the example the flight information with origin and destination. The data model behind this semantic representation is directly supplied with the attribute `emma:model-ref` that refers to the model defined in the `emma:model` tag at the end of the listing.

Further annotations are given as attributes within the element `emma:one-of` which are valid for all `emma:interpretation` elements it contains. They comprise annotations for modality classification (`emma:medium`, `emma:mode`), information about the raw input source (`emma:source`, `emma:signal`, `emma:signal-size`, `emma:process`, `emma:media-type`), the consulted grammar (`emma:lang`, `emma:grammar_ref`) and timestamps for the start and stop of the input (`emma:start`, `emma:stop`). Furthermore, it is possible to classify the inputs with respect to their communicative function (`emma:function`). EMMA also provides an extensibility mechanism for the annotation of user inputs with vendor or application specific metadata that is not covered by the standard. For this, the container `emma:info` is used. In the example the container introduces vendor specific session information.

If an interpretation instance is derived from another instance, the derivation annotation can be used to log the derivation process. Figure 3.15 shows the result of an interpretation process that goes from raw data to increasingly refined input representation. Here, interpretations from earlier stages of input processing are collected in the `emma:derivation` element. The first interpretation with the ID *raw* contains the result from the speech recogniser which is just the utterance of the speech input (*"from Boston to Denver tomorrow"*). The next interpretation with the ID *better* is result of the first interpretation step and describes flight information with the origin and destination of the flight and the unresolved specification *"tomorrow"* for the date. The `emma:derived-from` indicates that this interpretation is derived from the first interpretation with the ID *raw.* In the final interpretation with the ID *best*, the date is also resolved and the correct date for the flight is added. Here, the `emma:derived-from` indicates the direct derivation from the previous interpretation with the ID *better.*

3.3.3 SWEMMA

SmartWeb EMMA (SWEMMA) extends W3C EMMA and introduces additional structures for the representation of output (Sonntag and Romanelli, 2006). It was developed as part of the integrated ontology in the project SmartWeb (Oberle et al., 2007; Sonntag

```
<emma:emma version="1.0" xmlns:emma="http://www.w3.org/2003/04/emma" ...>
  <emma:derivation>
    <emma:interpretation id="raw"
      emma:medium="acoustic" emma:mode="voice">
      <answer>From Boston to Denver tomorrow</answer>
    </emma:interpretation>

    <emma:interpretation id="better"
        emma:process="http://example.com/mysemproc1.xml">
      <origin>Boston</origin>
      <destination>Denver</destination>
      <date>tomorrow</date>
      <emma:derived-from resource="#raw"/>
    </emma:interpretation>
  </emma:derivation>

  <emma:interpretation id="best"
      emma:process="http://example.com/mysemproc2.xml">
    <origin>Boston</origin>
    <destination>Denver</destination>
    <date>03152003</date>
    <emma:derived-from resource="#better"/>
  </emma:interpretation>
</emma:emma>
```

Figure 3.15 – EMMA derivation example (taken from Johnston et al. (2009))

et al., 2007). The idea was to model the result of output-related components (e.g., a presentation manager) in the extension, explicitly the results of the speech recognition analysis.

It contains concepts for monitoring the output generation process (`swemma:status`) and representing the result of the query in a processed interpretation. The result in the element `swemma:result` contains data for the output as well as the presentation. Since the project SmartWeb was focused on speech interaction, the output presentation was limited to speech synthesis. For the presentation of speech synthesis requests, the standard SSML was used that is nested inside of the element `swemma:result`.

3.3.4 SALT

SALT (Speech Application Language Tags) (Wang, 2002) is a spoken language interface standard from Microsoft Research for multimodal or speech-only applications. The main idea of the language is to adapt UI principles known from graphical web based applications to a design approach for spoken dialogues. This enables dialogue designers to concentrate on the pure user interface design instead of details in software engineering.

Wang compares a GUI application with a goal-oriented dialogue system where the information is exchanged with an iconic language. The interaction strategy is mostly system-initiative since the workflow of the interaction is normally prescribed by the design of the graphical user interfaces. Some mixed-initiative aspects can be realised using features such as type-in help or search boxes. The dialogue management is mainly page-based where the subtasks of a transaction are encapsulated in single pages or forms.

The SALT approach is built on the belief that spoken dialogues can also be modelled by a page-based interaction where each spoken interaction turn (the equivalent of a page) is designed to achieve a task's sub-goal. Thus, the same mechanisms that dynamically generate web pages today should be used for the planning of a speech-based dialogue. This includes the synthesis of spoken utterances and generation of grammars that define the utterances supported by a speech recognition component.

Consequently the object-oriented, event-driven model approach that is commonly used for GUIs is utilised for the integration of multiple input methods. If a user interacts with a GUI element, e.g., by a mouse-click on a graphical icon, this event is represented by an event object instance that contains all necessary information about the event, in this case the action type *mouse-click* and the coordinates of the mouse pointer on the screen. A speech input is correspondingly represented with the notion of semantic objects that capture the meaning of the spoken language. The semantic objects used are based on a type hierarchy.

Furthermore, SALT follows a semantic driven multimodal integration strategy and for this provides handlers that enrich the previously mentioned GUI events with a semantic representation of the objects with which the user interacts. This can be used for the resolution of cross-modal references, e.g., in a unification-based multimodal integration algorithm. Since speech objects share the same high level abstraction as GUI objects, in SALT speech objects can consistently interoperate with GUI objects, allowing multimodal dialogue designers to extend the currently used modality for communication.

The syntax for the SALT object examples is expressed in XML. The following snippet instantiates a SALT object that describes a speech input event:

```
<listen id="foo" onreco="f()" onnoreco="g()" mode="automatic">
  <grammar name="main" src="../meeting.xml"/>
</listen>
```

The *listen* instance defines the grammar used for speech recognition by a universal resource indicator (URI). Furthermore, the attributes *onreco* and *onnoreco* specify the callback methods that are executed in case of a successful respectively failed speech recognition event.

A SALT speech output object is a *prompt* object. It is used to perform text to speech synthesis (TTS) or to play prerecorded audio. A *prompt* request can contain static but also dynamic content by adopting a simple template-based approach for prompt generation. The following snippet shows an example. Here, the *value* tags refer to data from the input part of the SALT interaction turn:

```
<input name="origin" type="text"/>
<input name="destination" type="text"/>
<input name="date" type="text"/>
...
<prompt...>Do you want to fly from
  <value targetElement="origin"/> to
  <value targetElement="destination"/> on
  <value targetElement="date"/>?
</prompt>
```

SALT also supports the expression of deictic (or spatial) references within a *listen* object. The following snippet is part of a grammar that additionally to the supported utterances defines *propname* and *propvalue* attributes that are used to generate a semantic object:

```
<rule propname="drink" ...>
  <option> the </option>
  <list> <phrase propvalue="coffee"> left </phrase>
         <phrase propvalue="juice"> right </phrase> </list>
  <option> one </option>
</rule>
```

Dialogue designers can use this mechanism to bind particular speech phrases that contain spatial references to the current situational context, in this example the presentation of the two drinks on the GUI. Alternatively, the GUI may render a list of drinks and the speech grammar is adapted with the expressions *"the first one"* or *"the bottom one"*. In this approach the grammar is statically bound to the actual presentation in the GUI which means that a change in the GUI implicitly results in the demand of a change in the speech recognition grammar.

Unfortunately the integration of other modalities beyond GUIs and speech is not intended. The inclusion of speech act notions and pragmatics and natural language generation are planned as future work.

3.3.5 Summary and Conclusion

In this section four languages are presented for the representation of multimodal interaction. The following important features have been identified:

XML-based Language - All languages use XML-based languages for representation.

Continuous Refinement of Content - Since several components may deploy content to a message, it must be possible to continuously refine the message during the data workflow from raw data to interpretation.

Concrete Schemes for Modalities - Predefined schemes for the syntactic description of input and output should exist.

Modality Independent Representation of Interaction - The system should support a concept for the modality independent representation of content.

Semantic Knowledge Representation - Semantic knowledge representation should be supported based on ontologies.

Representation of Uncertainties - Recognition and interpretation uncertainties can be annotated with confidence values. This includes the representation of scope ambiguities. Scope ambiguities occur if two quantifiers take scope over each other and thus allow one to interpret an expression in different ways (Hurum, 1988). Humans are also capable of representing semantic ambiguity by means of underspecified representations that do not require all aspects of interpretation to be resolved (Poesio, 1994). An intermediate representation like the Quasi Logical Form (QLF) (Alshawi, 1990) is needed that contains unresolved terms corresponding to the uncertainties in underspecified representations.

Extensibility - The model should be extendable with content from new devices or interpreters.

Table 3.2 gives an overview of the features that are supported by the individual modelling languages introduced in this section. SALT already contains most of the above listed features but is designed for speech interaction in combination with GUIs, an extension with other modalities is not intended. The M3L language contains a large set of concrete representation schemes but is not designed for extensibility. EMMA and SWEMMA, which is based on ontologies and extends EMMA with output representations, are well elaborated and the only missing feature is a comprehensive approach for the uniform syntactical and semantical representation of interaction. This additional issue is addressed by SiAM-dp which is described in Chapter 5.

3.4 Dialogue Act Annotation

In order to successfully build a conversational model, the simple description of conversational behaviour is not sufficient. Even if the content of two conversations is equal, the actual set of exhibited behaviours differs from person to person and conversation to conversation (Cassell, 2000). This also means that one particular behaviour can be employed in a variety of circumstances to produce distinct communicative effects and the same communicative function may be realised through various sets of behaviour. The

	Input	Output	Multimodal	XML-based	Cont. Refinement	Concrete Schemes	Modality Independent Representation	Semantic Knowledge Representation	Representation of Uncertainties	Extensibility
M3L	x	x	x	x	x	x	x	x	x	
EMMA	x		x	x	x		x	x	x	x
SWEMMA	x	x	x	x	x		x	x	x	x
SALT	x	x	o	x		x	x	x	x	
SiAM-dp	x	x	x	x	x	x	x	x	x	x

Table 3.2 – Features supported by the individual interaction modelling languages. The symbol (o) indicates that multimodality is restricted to speech interaction and GUIs.

actually applied communicative behaviour is dependent on the type of conversation, availability of modality, cultural patterns, and also the personal style.

Thus, in order to build a model for the description of how conversation works, it is necessary to identify high level structural elements of interaction and their communicative function in the discourse. These elements describe a role or function in a contribution, e.g., *conversation invitation*, *turn taking*, *providing feedback*, or *task requests* (Cassell, 2000). Especially the examination and annotation of linguistic data at the semantic level in terms of their communicative function experience a growing interest resulting in various dialogue act annotation schemes. The maybe most widely used one is DAMSL (Dialogue Act Markup using Several Layers) by Allen and Core (1997).

The scheme DIT^{++} (Bunt, 2009) extends this scheme with the possibility to annotate multifunctional communicative acts, which is necessary because of the phenomenon that utterances in a dialogue often have more than one communicative function (Bunt, 2011b). For example, the request *"Henry, could you take us through these slides?"* contains, besides the function to assign the turn to Henry, also a specific request. It is argued that this phenomenon can be explained by analysing the participation in a dialogue as involving the performance of several types of activities in parallel, relating to the different dimensions of communication (Bunt, 2011a). Therefore the annotation language of dialogue acts must be multidimensional which means that multiple tags must be assignable to a single dialogue act (Petukhova, 2011).

Since human communication is not only restricted to verbal communication, also non-verbal and multimodal dialogues can be annotated with these schemes. Petukhova and

Bunt (2012) argue that nonverbal behaviour may emphasize the intended meaning of synchronous verbal behaviour. It may also express a separate dialogue act in parallel to a verbal speech act expressed by the same speaker and add multi-functionality to a communicative contribution.

Petukhova and Bunt (2012) also point out the importance of a distinction between the description of the conversational behaviour and the communicative function, more precisely the separation from description and interpretation (in this thesis we will address this issue also with a distinction between the syntactic and semantic representation of a dialogue act). This is well-founded by the argument that even in speech-only dialogues one function can be expressed in several different ways. Above all, the same functions can additionally be communicated by diverse modalities.

A model that decouples the two abstraction levels of representation easily allows one to consider the above mentioned phenomena. The following subsections will discuss how three frameworks for multimodal interaction support and realise this idea. The fourth related work presented is an ISO standard for the annotation of dialogue acts on which the dialogue act model in SiAM-dp is based.

3.4.1 EMMA specification

EMMA (see also Section 3.3.2) does not directly support a well elaborated annotation language for the description of communicative functions. Anyway, the standard is aware of the fact that a user input can serve as several functions. Thus, orthogonally to the mode, user inputs can be classified with respect to it. The simple classification is defined by the attribute `emma:function` that contains a string with a value of the open set {`recording, transcription, dialog, verification, ...`}. A more concrete subdivision is not specified by the standard.

3.4.2 SAIBA framework

SAIBA is a framework for the multimodal generation of communicative behaviour with embodied conversational agents (Kopp et al., 2006) with the main goal to help researchers in building more sophisticated virtual humans. It is based on a three stage model called SAIBA (Situation, Agent, Intention, Behaviour, Animation) where the stages represent intent planning, behaviour planning and behaviour realisation. Figure 3.16 depicts the three different levels of the generation process. This approach is necessary since the generation of natural multimodal output requires a time-critical production process with high flexibility.

The framework defines two markup languages that serve as interfaces between the generation steps: FML and BML.

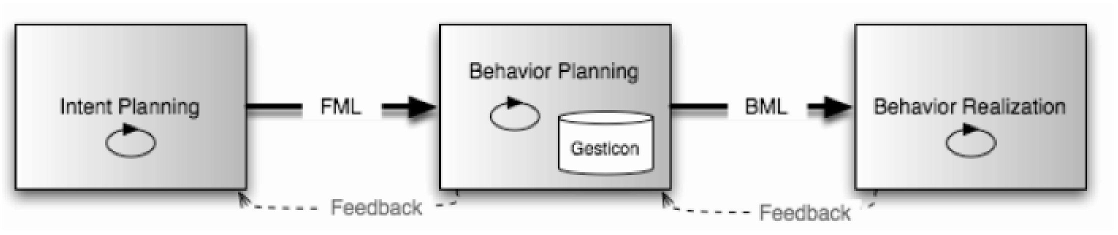


Figure 3.16 – Steps during multimodal output generation in the SAIBA framework (from Kopp et al. (2006))

The *Function Markup Language* (FML) describes the communicative function at the level of intent independent from the actual realisation of the surface behaviour. It constitutes the interface between the *Intent Planning* and *Behaviour Planning* components and provides a semantic description of a dialogue act and its associated semantic units or content. A unified language for the functions is still a work in progress but the containing aspects are relevant for the planning of verbal and nonverbal behaviour.

The *Behaviour Markup Language* (BML) describes the surface realisation of multimodal behaviour which is then implemented by realisation engines like GUI or speech renderers. The latter can theoretically realise every aspect of content the behaviour planner is generating (verbal, gestural, phonological, etc.). In practice, the scope is often restricted to a limited set of predefined realisations. However, BML avoids specific process implementations and provides general descriptions of multimodal behaviour independent from the applied renderer.

Example

Vilhjálmsson (2009) provides an example that gives a better understanding of the importance of the distinction between communicative function and communicative behaviour (see Figure 3.17).

Here a narrator wants to report about his contrasting chocolate flavours. The communicative intent is represented on the left side of the figure in a formal notation. It contains the wish to open a new topic, a representation of emotions and finally the communicative function to inform the listener about the flavours. The figure demonstrates two actually applied conversational realisations for the narrator's intent. In the first case, the recipient of the message is not present and the the realisation of the communicative intent is written. In the second case the recipient is present and the intent is realised in a combination of spoken language and body gestures.

The example points out that the actual realisation of communicative behaviour is highly dependent on the situational context, here the presence of the listener. The communicative intention, however, stays the same. Thus, the division into two levels of abstraction helps modern applications to integrate multimodal behaviour.

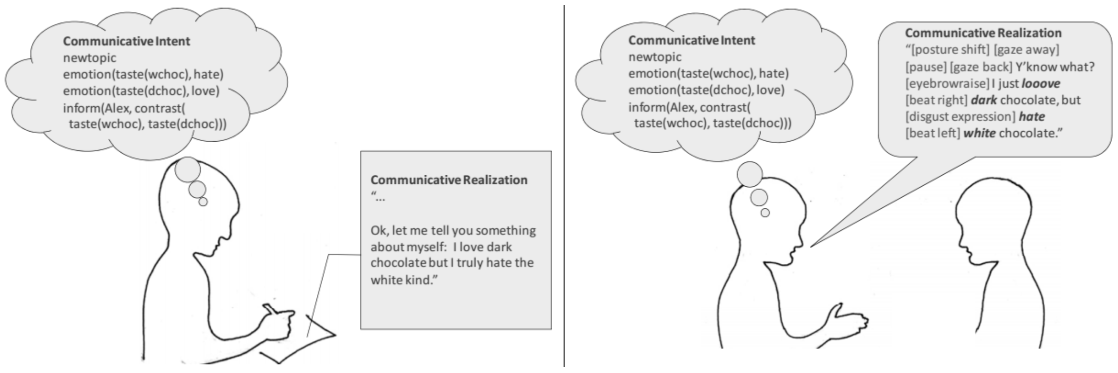

Communicative Intent
newtopic
emotion(taste(wchoc), hate)
emotion(taste(dchoc), love)
inform(Alex, contrast(
taste(wchoc), taste(dchoc)))
Communicative Realization
"...
Ok, let me tell you something about myself: I love dark chocolate but I truly hate the white kind."
Communicative Intent
newtopic
emotion(taste(wchoc), hate)
emotion(taste(dchoc), love)
inform(Alex, contrast(
taste(wchoc), taste(dchoc)))
Communicative Realization
"[posture shift] [gaze away] [pause] [gaze back] Y'know what? [eyebrowraise] I just looove [beat right] dark chocolate, but [disgust expression] hate [beat left] white chocolate."

Figure 3.17 – Two different and situation dependent forms for the realisation of a communicative intent (from Vilhjálmsson (2009))

Especially in an embodied conversational agent architecture (see Figure 3.18), the central dialogue module can make decisions purely on the abstract level of communicative intention. The modules that understand or generate the behaviour map the agent's intentions to behaviour and vice versa, depending on the current context. This concept enormously simplifies the process of adapting an agent's behaviour to particular situational contexts like users, cultures, or available devices.

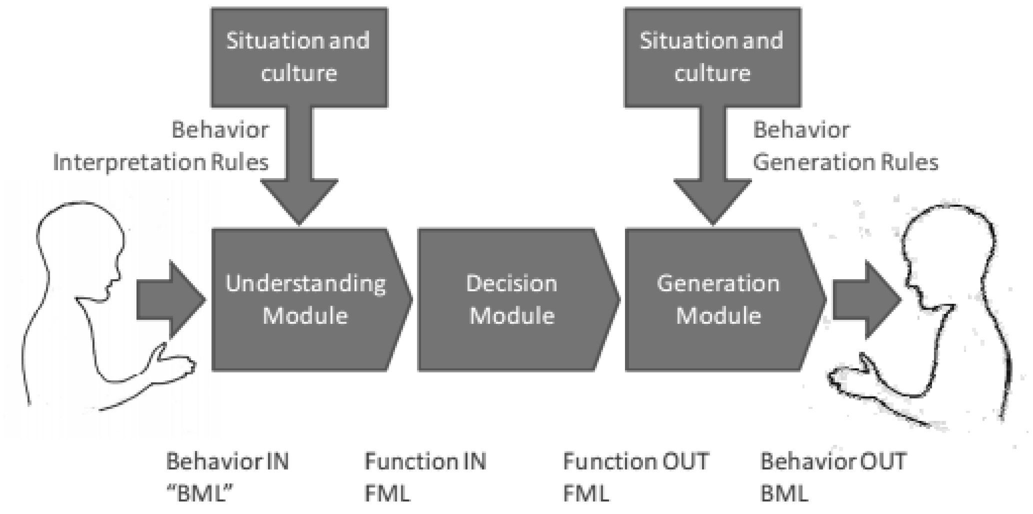


Figure 3.18 – Example for a conversational agent architecture where decisions are only made on an abstract representation of intentions (from Vilhjálmsson (2009))

3.4.3 CDE framework in VirtualHuman

The VirtualHuman project (Reithinger et al., 2006) aimed at developing a scenario with virtual characters that interact in cooperation with the user as comprehensive, life-like dialogue partners. In the demonstrator system (Figure 3.19), the user can multimodally communicate with virtual characters that are located in a highly realistic virtual environment. A main focus was placed on the simulation of lifelike interactivity in a mixed group of real people and believable virtual characters. Thus, characters show the typical interaction behaviour that is expected by real humans depending on the actual situation of interaction (Löckelt et al., 2007). The 3D rendering engine creates naturalistic virtual characters with the ability to show communicative behaviour using body gestures and facial expressions and can even express emotions, moods and personality. These are combined with spoken utterances that are generated by a text-to-speech system and synchronised with the non-verbal behaviour.

The communicative behaviour of the virtual characters is represented with the Player Markup Language (PML), a XML-based language that provides both high level abstract concepts (e.g., gestures, complexions, emotions) and detailed, technical information required for the character and player-related realisation of those concepts, e.g., timing information. One important part of this language are PML actions that are used for the specification of synchronised multimodal output (e.g., postures, gestures, facial animations, speech). This representation of interaction is located on the level of surface realisation.

For each of the dialogue participants, the real and virtual humans, a conversational dialogue engine (CDE) is responsible for dialogue and behaviour control. The atomic units of the communication are communicative acts, the set of possible communicative acts is shared and agreed upon between all CDEs. They are organised in an ontology

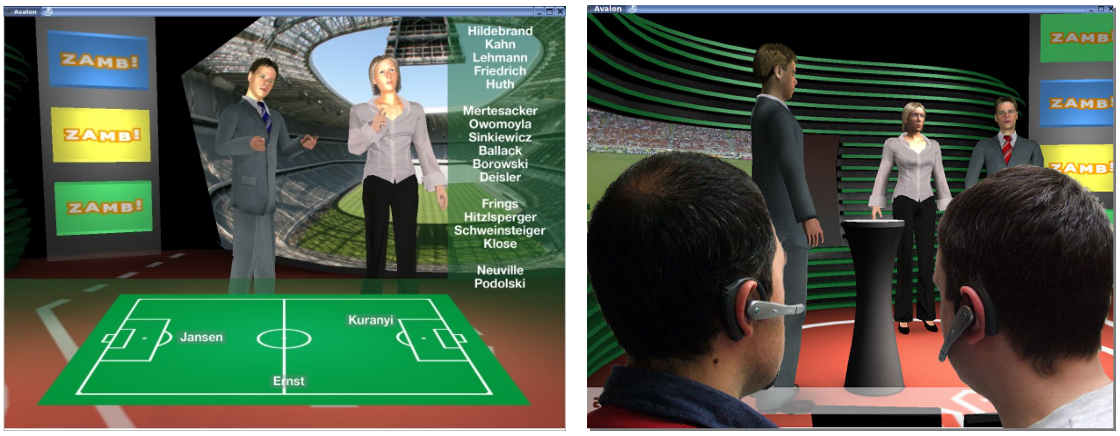
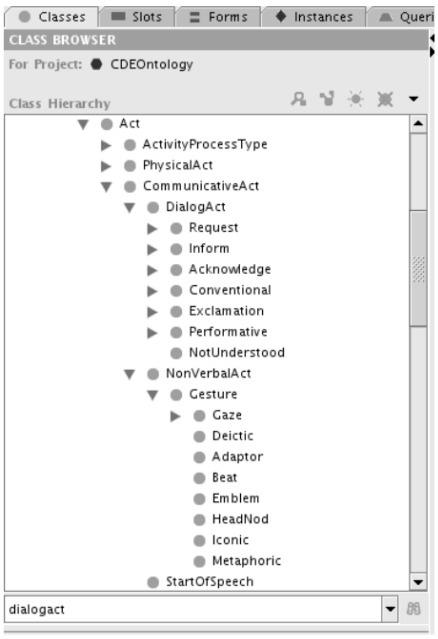


Figure 3.19 – The VirtualHuman demonstrator system (from Reithinger et al. (2006))

and extend concepts of the common DAMSL annotation scheme (Allen and Core, 1997). The top-level class of the hierarchy is the *Act* class which is parent of four kinds of acts: *PhysicalActs*, *DialogActs*, *NonverbalActs*, and *StartOfSpeech*. Figure 3.20 shows an excerpt of the hierarchy. *NonverbalActs* include gestures like *Iconics*, *Metaphorics*, and *Gazes*. Instances of the *DialogueAct* class represent the intention of a character and are further divided into more concrete concepts, e.g., *Questions*, *Responses* and *Statements* (Kempe et al., 2005).

Pfleger (2007) points out that the VirtualHuman architecture distinguishes between User-CDEs and Character-CDEs. While the User-CDE represents a human user of the system, the Character-CDE represents an autonomous character of the virtual world. Accordingly the User-CDE is more responsible for input processing and has to convert the recognised verbal and nonverbal actions by the user into instances of the above mentioned ontology, applying speech recogniser, gesture recogniser and natural language understanding components. On the other hand, the Character-CDEs generate multimodal output in the form of synchronised verbal and nonverbal contributions of the virtual character. For this the ontological instance of a dialogue act is converted into the surface realising PML syntax, containing spoken utterances and nonverbal actions.

Overall, it can be said that the CDEs serve as a translator between an abstract ontological instance of a dialogue act and the multimodal contribution of a participant. While the surface realisation is appropriate for the interaction with the user, internally the interfaces between the diverse CDEs and dialogue management are based on the abstract dialogue act representations.



For Project: CDEOntology

Class Hierarchy

```
Act
  ActivityProcessType
  PhysicalAct
  CommunicativeAct
    DialogAct
      Request
      Inform
      Acknowledge
      Conventional
      Exclamation
      Performative
      NotUnderstood
    NonVerbalAct
      Gesture
        Gaze
        Deictic
        Adaptor
        Beat
        Emblem
        HeadNod
        Iconic
        Metaphoric
    StartOfSpeech
```

dialogact

Figure 3.20 – Overview of the Act hierarchy in the CDE ontology (from Pfleger (2007))

3.4.4 Semantic Dialogue Annotation Framework ISO 24617-2

ISO 24617-2:2012 (2012) is an international standard for the annotation of dialogues with dialogue act information (Bunt, 2011b; Bunt et al., 2010). The motivation for the standard was a wide range of alternative dialogue act taxonomies and inventories causing considerable terminological and conceptual confusion and incompatibility of annotated corpora (Bunt, 2011b). The standard is partly based on the DIT^{++} (Bunt, 2009) taxonomy which follows a dynamic approach for describing utterance meaning in the Dynamic Interpretation Theory (DIT). In this theory dialogue acts correspond to update operations on the information states of participants in the dialogue.

The representation of dialogue acts is defined in the Dialogue Act Markup Language (DiAML), a formal language for expressing dialogue annotations. With the dialogue act theory it follows an empirically-based approach to the computational modelling of communication, in particular of linguistic and nonverbal communicative behaviour in dialogue. Furthermore, the standard provides a set of empirically and theoretically well-motivated concepts for dialogue annotation and a method for segmenting a dialogue into semantic units.

The standard defines a dialogue act as *"a semantic unit of communicative behaviour in dialogue, which has a certain communicative function (possibly more than one) and semantic content"*. Figure 3.21 depicts how dialogue act annotation concepts are related. A dialogue is divided into two or more functional segments. Each functional segment is related to one or more dialogue acts, allowing one to represent a possible multi-functionality of functional segments. Nevertheless, the sender of a functional segment is exactly one dialogue participant, the addressee can be one or also more participants if the sender communicates with a group. Furthermore, participants in other roles can be modelled, so-called side-participants. Semantically, a dialogue act is divided into a communicative function part and a semantic content category. While the latter provides information about the type of semantic content carried by the dialogue act, the former indicates how the particular function of the sender's behaviour should be understood.

The annotation scheme aims to support both manual and automatic annotation. Thus, the concepts can be settled at various granularities: concepts with a depth that match human understanding of a communicative function and additionally more rough concepts that allow a more surface-oriented form of annotation. For this, the standard defines a hierarchy of communicative functions where functions deeper down in the hierarchy are more fine-granular. It contains two different types of functions:

Dimension-specific functions
These functions often do not transport semantic content. They are used to annotate utterances that are responsible for *turn management*, *providing feedback*, *time management*, *discourse structuring*, *communication management*, and *social obligations management.*

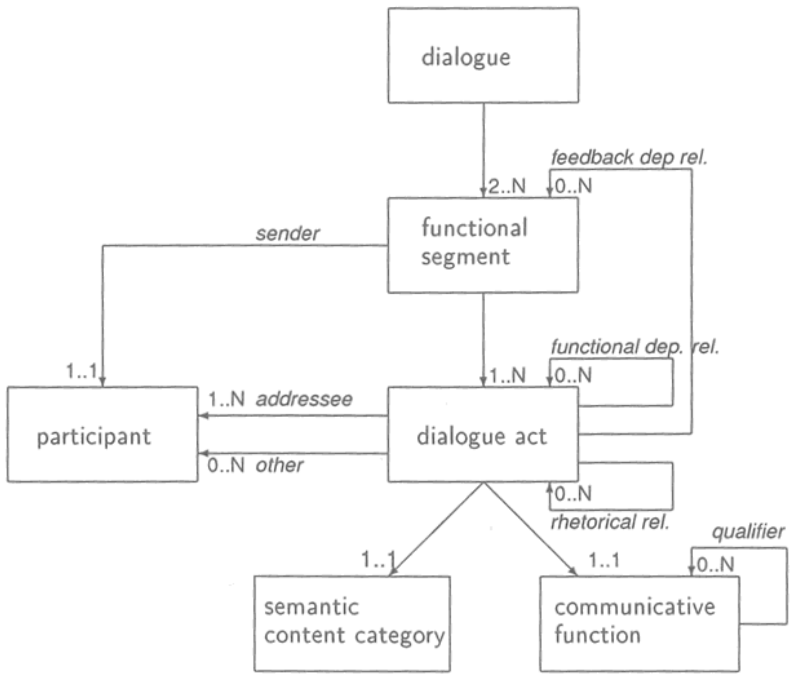


Figure 3.21 – Meta-model for dialogue act annotation (from ISO 24617-2:2012 (2012))

General-purpose functions
These are domain-independent functions that transfer information or discuss communicative or other actions. Figure 3.22 depicts this hierarchy. It contains two main groups: *Information-transfer functions* and *action-discussion functions*. The first type can again be subdivided in *information-seeking functions* and *information-providing functions*. The latter is split into the main groups *commissives* and *directives*. Farther downward in the hierarchy tree, the functions become more concrete and are used to annotate functions like *check question*, *confirm*, *accept offers*, or *decline requests*. General-purpose functions often transport semantic content, e.g., the information that is provided/requested or a specific task that is content of an instruction.

The ISO standard explicitly claims that the core annotation concepts provided cannot be expected to be ideal for every kind of dialogue analysis, for every task domain, every dialogue, or every annotation purpose. Hence, it is open and provides guidelines and general principles for extensions (e.g., domain-specific concepts), and for selecting coherent subsets of core concepts.

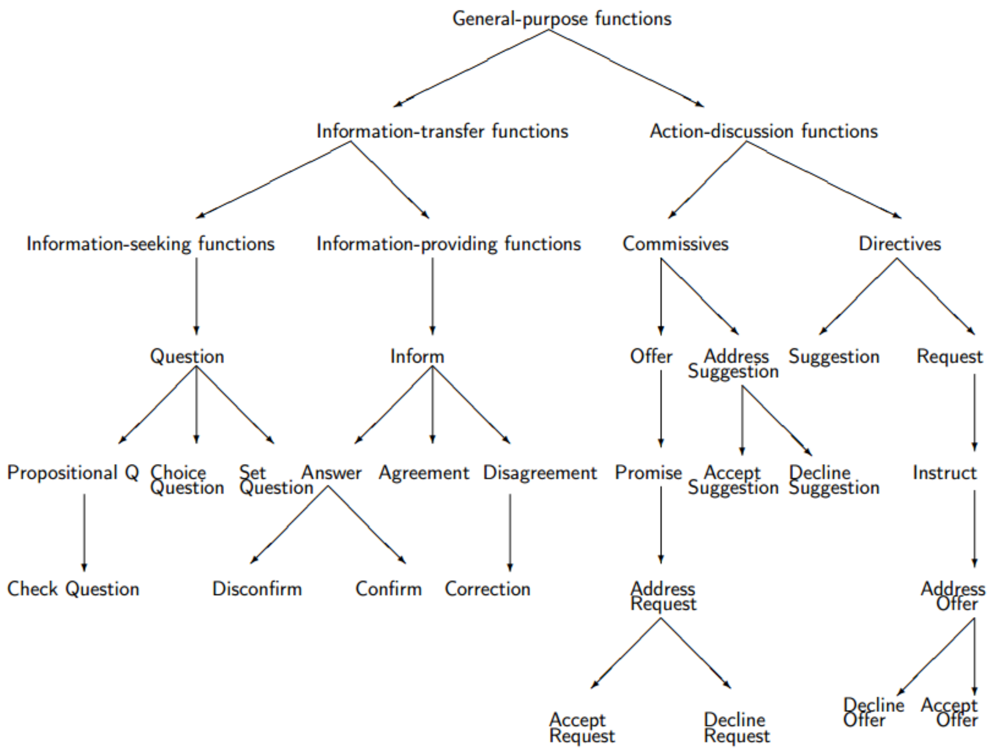


Figure 3.22 – Hierarchy of communicative functions in ISO 24617-2:2012 (2012)

3.4.5 Summary

This section discussed the advantages of semantic dialogue act annotation. Several projects were examined that consider this way of representing dialogue acts in HCI. We identified the following relevant aspects that influence the work presented in this thesis:

Separate representation of communicative behaviour and function

It has been discussed that the communicative behaviour of a dialogue participant can be interpreted in various ways, depending on cultural, personal, and situational issues. Thus, the interpretation or intention behind an extrinsic identical verbal or nonverbal communicative act can differ. Furthermore, one and the same communicative intention can be expressed in different ways. There can be variations in the utterances of spoken language or even non-verbal acts expressing the same intention.

In order to consider the loose coupling between communicative function and behaviour, it makes sense to provide representation languages for both of them. Furthermore, a mechanism for the transformation between both models is necessary.

On the input side *interpreters* have the task of understanding the intention of a dialogue participant's behaviour and representing it in the language for communicative functions. On the output side the system's communicative intention must be presented to the dialogue participant in a way that is perceptible for him. For this output *generators* or *renderers* are responsible for, e.g., synthesising spoken language or generating graphical user interfaces.

Dialogue Management based on communicative functions

The frameworks presented use communicative functions for the internal planning of dialogue management decisions and for the communication between involved modules. This has two advantages:

First, the decisions in dialogue planning can be made purely on the more abstract representation of the dialogue participants' intentions. The actually consulted modalities and surface realisations must not be regarded at this point. This automatically makes the dialogue management independent from the effectively used modalities and devices. Thus, a compensating or complementing extension with new devices is realisable without adapting the dialogue management specification of an application.

Second, the dialogue applications become highly flexible in terms of the devices and surface realisation actually used. These can be created by generators or renderers, regarding situational, personal, or cultural aspects.

Classification of communicative functions in a hierarchical taxonomy

Some of the dialogue act annotation languages presented use a hierarchical taxonomy for the classification of dialogue acts. This allows one either to build more precise annotations with concepts from a deeper hierarchy level, or less fine-grained annotations with communicative functions from a higher level. Thus, on the one hand it is possible to make manual annotations with a satisfying reliability, on the other hand automatically performed annotations maintain a high accuracy.

Table 3.3 gives an overview of the multimodal dialogue-act annotation aspects that have been addressed by the individual related work presented in this section. The first three columns classify the work into international standards, language specifications, and dialogue systems. The other columns recall the above mentioned aspects and depict which of the presented languages support them.

The first three annotation languages for dialogue acts have not primarily been designed for the use in a dialogue system. Nevertheless, their concepts are well suited to describing the intention of an interaction act in the dialogue system and the hierarchy for communicative functions is a feature that has not been deeply elaborated on in the other related work presented in this subsection. This work fills this gap by adopting these concepts, which are even part of an international standard, and using them for the modality independent representation of interactions.

Both presented frameworks SAIBA and CDE show how important the separation between the representation of the communicative behaviour and the intention is in order to build highly flexible multimodal dialogue systems. Both systems use interpreters and generators for the translation between behaviour and intention. SiAM-dp transfers these ideas and provides models for describing interactions on both levels of abstraction and strategies for realising the translations between them (see Chapter 5). In contrast to the other dialogue system frameworks presented in this section, the description of the intention is based on the concepts of international dialogue act annotation standards. This approach is assumed to provide a good basis for further research in dialogue management as well as speech and language processing in combination with a diversity of additional modalities.

	International Standard	Language Specification	DS Framework	Representation of Behaviour	Representation of Intention	Behaviour Interpreter	Behaviour Generator	Situational Adaptive Behaviour	Hierarchy for Communicative Functions
DAMSL		x			x				
DIT++		x			x				x
DiAML / ISO 24617-2	x	x			x				x
EMMA	x	x		x	o				
SAIBA			x	x	x	x	x	x	
CDE / Virtual Human			x	x	x	x	x		o
SiAM-dp			x	x	x	x	x	x	x

Table 3.3 – Identified dialogue act annotation aspects in related work. The symbol (o) indicates that the aspect has been considered but not deeply elaborated upon.

4 The SiAM-dp modelling language

The choice of the right modelling language has been the first and a very important step for the design of the model-based development approach in this thesis and a lot of effort has been put into this question. Several aspects from diverse requirements have influenced the final decision, outgoing from the findings in related work and current languages and formats for representing meaning and metadata in multimodal dialogue systems. They mainly raise demands on the expressiveness and the representation of meaning and knowledge as well as reasoning capabilities for the fusion of information from different sources. These aspects are analysed in Sections 4.1 and 4.2.

Furthermore, several practical issues have been taken into account since the modelling language must form the basis of the complete dialogue platform, including the runtime components and a development environment. Section 4.3 discusses these requirements and presents the modelling language finally used in the Situation Adaptive Multimodal Dialogue Platform (SiAM-dp) and the extensions that have been developed in order to fulfill them. Section 4.4 presents a model for defining match patterns which is a valuable feature throughout the dialogue platform.

4.1 Semantic Knowledge Representation

The early multimodal systems presented in Section 3.1 already used knowledge bases for fusion and reference resolution. They mostly accessed taxonomies and typed feature structures for the representation of their knowledge bases. Milward and Beveridge (2003) elaborate that semantic knowledge and ontologies may play a relevant role in the replacement of hand crafted dialogue design with generic dialogue systems that work on an ontological domain knowledge. Thus, the complexity of dialogue system components could be reduced. In their case studies that worked on speech-based dialogue systems, they showed that, additionally to reference resolution, ontologies can contribute to dialogue management, speech recognition, speech interpretation, and speech generation. Other state-of-the-art dialogue platforms that rely on ontologies were introduced in Section 3.2. Here the semantic representation of knowledge is especially exploited in discourse and context resolution in order to gain relevant knowledge about a user's intention in the context of the actual discourse and environment.

A base requirement for those systems is a suitable meta-language that can represent the content of interactions in a semantic and thus machine-understandable way. It is necessary to have a structural framework that allows one to organise the information, to define concepts and to set them in relation to each other. Several W3C and semantic web projects developed standards for knowledge modelling, e.g., the Resource Description Framework (RDF), RDFs, DAML, and OWL. Others try to annotate information that is available in human-readable form with additional semantic content. RDFa, for instance, allows one to embed rich metadata within already existing HTML, XHTML or XML-based web documents, making the information presented in this document additionally machine-understandable.

All these languages can be grouped under the term *ontology language*. Originally the term ontology comes from philosophy and is the study of *'what there is'*, i.e., whether a certain thing, or more broadly entity, exists (Hofweber, 2014). Furthermore, it describes the most general features and relations of entities. With an ontology language it is possible to set a syntactic predication of formal logic into relation to an interpretation with model-theoretic semantics. A difference is made between the *terminological component* (TBox) and the *assertion component* (ABox) of an ontology. The TBox describes the terminology of a world with a set of concepts, their relations, and their properties. The ABox describes concrete objects (individuals) within a knowledge base by TBox-compliant statements.

In computer science, ontologies play a relevant role for knowledge engineers (ontologists) and researchers in artificial intelligence. The following epistemological primitives are the most common for ontologies and are supported by most semantic modelling languages:

Concepts: Ontologies contain taxonomies that group entities into *concepts* (also called *classes*) that share common features (e.g., attributes or relations). For example the concept *Vehicle* represents all classes of vehicles like cars, airplanes, or bikes. Often the taxonomies have a hierarchical structure and define derivations by *is-subtypeOf* relations.

Attributes: Each concept in an ontology can contain attributes that are defined by a name or identifier and an attribute type. They describe a special characteristic or feature of a concept, e.g., the colour of a car.

Relations: Relations build up connections between the concepts of an ontology. Mostly the relations *is-a* and *is-subtypeOf* relations are used in order to define individuals of a concept or hierarchies between concepts. Another common representative is the *has-a* relationship.

Individuals: Individuals are instantiations of a specific concept and defined by the *is-a* relation. For example, the individual Angela Merkel is an instance of the concept *Person*.

In the following subsections some common semantic modelling languages are presented (see also Allemang and Hendler (2008)).

4.1.1 RDF - Resource Description Framework

A basic framework of the semantic web as promoted by W3C is the Resource Description Framework (RDF)[1]. In the semantic web things or entities are referred to as *resources* that can be directly identified by a Uniform Resource Identifier (URI). Nevertheless, the availability of the resource at the URI is not absolutely necessary. RDF has been developed for the annotation of these web resources with meta-information. The basic building blocks of RDF are the so-called subject-predicate-object triples. Here the subject and object are set in relation by a predicate. Usually RDF models are presented as XML-document (Figure 4.1 shows an example). RDF is the basis for the more expressive languages RDFs and OWL.

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">
<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire␣Burlesque">
  <cd:artist>Bob Dylan</cd:artist>
  <cd:country>USA</cd:country>
  <cd:company>Columbia</cd:company>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>
</rdf:RDF>
```

Figure 4.1 – Example of an RDF instance describing a music CD from Bob Dylan (code snippet taken from w3schools.com)

4.1.2 RDF Schema

The Resource Description Framework Schema (RDFS)[2] is a recommendation from W3C for the representation of knowledge. It builds upon RDF with the intention to structure resources. For this it provides basic concepts for the description of ontologies. Practically, the schema is an extension of the RDF vocabulary with - inter alia - the following additional epistemological primitives:

[1]http://www.w3.org/RDF/

[2]http://www.w3.org/TR/rdf-schema/

- **rdfs:Class:** Declares a concept/class resource for other resources
- **rdfs:datatype:** A class for datatypes. rdfs:Datatype is a subclass of rdfs:Class
- **rdfs:subClassOf:** Predicate that declares hierarchies of classes.
- **rdfs:subPropertyOf:** Declares relations between two properties that are used as predicates.
- **rdfs:domain:** Declares the class of a property's subject.
- **rdfs:range:** Declares the class of a property's object.

Figure 4.2 shows a RDFS snippet that describes the concept *Animal* and its subconcept *Horse*.

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdf:Description rdf:ID="animal">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="horse">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#animal"/>
</rdf:Description>

</rdf:RDF>
```

Figure 4.2 – Example of an RDFS document. It defines the concept animal and a subclass horse. (code snippet taken from w3schools.com)

4.1.3 OWL

The Web Ontology Language (OWL)[3] enables knowledge engineers to design ontologies and knowledge bases. The language was majorly influenced by the ontology language DAML+OIL and extends the RDF framework. The specification of OWL is endorsed by W3C. Several sublanguages like OWL Lite, OWL DL, and OWL Full allow different levels of expressiveness. OWL provides constructs for the expression of constraints and restrictions. In a simple way these are cardinality constraints but the more expressive OWL DL and OWL Full also support description logic that gives ontologies a meaning.

[3]http://www.w3.org/TR/owl2-syntax/

4.2 Typed Feature Structures

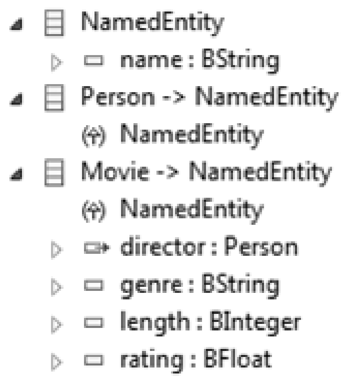
Typed Feature Structures (TFSs) provide a different way to represent complex structured data than using triples like RDF does. TFSs extend the concept of feature structures that are sets of slot-value pairs (Carpenter, 1992). The slots are identified by names and contain values that can either be atomic or complex. A complex value can again be a feature structure or a collection of several entries. Feature structures can alternatively be seen as a directed acyclic graph. In this case the nodes are the values that are connected by paths that represent the attributes.

The added value of typed feature structures in comparison to feature structures is that they are based on type hierarchies. This allows one to restrict the permitted slot-fillers of a slot to a specific type and its subtypes. Furthermore, a subtype inherits all slot definitions from its supertype.

Figure 4.3a shows a type hierarchy. The concepts *Person* and *Movie* are subtypes of the concept *NamedEntity* and inherit the slot *name* with the atomic datatype *BString.* The concept *Movie* contains three slots with atomic datatypes as slot-fillers (*genre, length, rating*). The slot director has the complex type *Person* as a slot-filler. Figure 4.3b shows the TFS instance *Iron Man 3* which is of the concept *Movie.*

TFSs have often been used in Natural Language Processing (NLP) for the representation of lexical entries, grammar rules, and communicative meaning (Oviatt et al., 2000). Krieger and Schäfer (1994) list some benefits of TFSs, e.g., structured knowledge, efficient processing, type checking, and recursive types. A further advantage is that unification and overlay operations allow one to compare and merge information from two typed feature structures to one common representation (see also Section 4.3.3).

TFSs are also important for the modality fusion in multimodal dialogue systems. One of the first papers where TFSs are mentioned together with multimodal fusion in dialogue



```
NamedEntity
    name : BString
Person -> NamedEntity
    NamedEntity
Movie -> NamedEntity
    NamedEntity
    director : Person
    genre : BString
    length : BInteger
    rating : BFloat
```

(a) Type hierarchy.

$$\begin{bmatrix} \textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{director:} & \begin{bmatrix} \textbf{Person} & \\ \text{name:} & \text{Shane Black} \end{bmatrix} \\ \text{genre:} & \text{Action} \\ \text{length:} & 130 \\ \text{rating:} & 4.5 \end{bmatrix}$$

(b) TFS instance.

Figure 4.3 – Example of a typed feature structure

systems is Cohen et al. (1997). Various types of unification and overlay algorithms on TFS were also involved in the SmartKom project (Wahlster, 2002) and constituted the fundamental computational processes for modality fusion and fission.

4.2.1 Extended Typed Feature Structure

Pfleger (2007) introduced *extended Typed Feature Structures (eTFSs)* for his *Ontology Based Dialogue Platform (ODP)*. The goal of his approach was to complement TFS with features known from RDF/RDFS. One of these extensions was the support of cardinality constraints for multiple slot-fillers. Thus, besides defining type restrictions for slot-fillers it is also possible to set the minimum and maximum cardinality of a slot's content (as it is possible in OWL).

A further extension was the support of unique identifiers that allow co-referenced objects across documents. With the unique identifier one instance can be the slot filler of two slots in different documents. Thus, changes to an instance in one document also affect the referred instance in the other document. In classical TFSs this support is only restricted to one document.

Furthermore, each eTFS instance can be annotated with an activation value that has a range between 0 and 1 and defines its accessibility from *not accessible* to *highest accessible*. Pfleger used this concept as a resource for computational processes in reference and discourse resolution (Pfleger and Alexandersson, 2006) and as an input parameter for the weighting component of *PATE*, a production rule system based on typed feature structures (Pfleger and Schehl, 2006).

Finally, he provided a well-defined Java API for the work on eTFS instances. The library, which implemented this API, supported operations for pattern matching and merging based on unification and overlay (see Section 4.3.3). Furthermore, access methods allowed one to traverse complex structures and to examine substructures for specific types.

It turned out that the features of eTFS are very useful for the development of multimodal dialogue systems. Thus, it had a great influence on the design of the modelling language for the SiAM dialogue platform. This modelling language is introduced in the next section.

4.3 SiAM Meta Model

4.3.1 Requirements

Gurevych et al. (2003) claim that a complex dialogue system with many distributed but cooperative components should utilise a uniform representation language which forms a common language between all involved components. Thus, it is easier to exchange information between modules, to reuse modules and to switch between different domains. Outgoing from this request, the first task for the development of the new multimodal dialogue system was to find an appropriate modelling language. Originally, the request from Gurevych concerned the representation of domain knowledge. In SiAM-dp we want to go a step further and model everything with only one meta-modelling language. This embraces the following aspects:

- **Domain Knowledge:** Both the terminology (TBox) of an ontology and its individuals (ABox), should be representable with the same modelling language. This idea is taken from RDFs where typification and instantiation are realised by entity-attribute-value triples and the RDF vocabulary.
- **Dialogue Platform Resources:** These are resources that are necessary for running the platform like session information, user description, device description and media resources.
- **Input/Output Interface:** The communication between the platform and external input and output devices as well as actuators and sensors is based on a markup language (see Section 5.3) that should be modelled with the common language. This model should also be used between internal core components that are involved in the main interaction workflow.
- **Grammar Rules for Speech Recognition:** For applications that support speech input, the platform should provide a common model for grammar rule specification (see 6.2).
- **Graphical User Interfaces:** Graphical presentation should be internally represented with a common model for graphical user interfaces (see Section 6.3).
- **Dialogue Workflow Specification:** The platform should provide a model for the declarative specification of the interaction workflow (see Section 6.1).
- **Dialogue Application Project Specification:** The platform allows one to declaratively create dialogue applications. This means a dialogue project will consist of several instances of the previously mentioned models. These instances are summarised in a project application specification which should also be represented with the same modelling language (see Section 6.4).

Another advantage of a common modelling language, besides the easy interchange of information between all platform components, is that instances from one model can be embedded into the instance of another model. For example, an instance of the knowledge model can be content of the model for defining input and output messages. This message can again be part of a model that describes the dialogue application's behaviour.

Findings from the previous section helped to identify the following requirements for the dialogue modelling language:

1. Meta-modelling language for defining domain models
2. Support of type hierarchies
3. Multiple inheritance
4. Type restriction for slots
5. Cardinality restriction for slots
6. Co-references between different documents
7. Support of pattern matching and merging algorithms based on unification and overlay

For practical reasons regarding the implementation and integration, additionally the following features are claimed:

1. Serialisation of model instances for the communication between the platform and external components
2. Generation of POJOs (Plain Old Java Objects) for platform internal representation of objects
3. Editor support for the creation of instances

All these requirements are fulfilled by the Eclipse Modeling Framework (EMF), which is introduced in the next subsection.

4.3.2 Eclipse Modelling Framework

EMF is an open-source Java framework for modelling that exploits the facilities provided by Eclipse. It was originally developed to create structured data models in model-driven software engineering (Steinberg et al., 2009). The core (meta-)model of EMF is *Ecore*, which is a reference implementation of EMOF (Essential Meta Object Facility), a standardised specification for meta-model architectures [4]. Ecore also is an EMF model and thus its own metamodel (a subset of the Ecore metamodel is shown in Figure 4.4). Consequently, not only domain models are expressed with Ecore but also the concepts that are used to describe the domain model. This fact makes it possible to reference model classes from inside a model instance. This will be especially important for the pattern definition as described in Section 4.4.

The semantic expressiveness is comparable to eTFS since it is possible to define type hierarchies, slots, and restrictions for the slot type as well as cardinalities.

The language does not provide the full semantic expressiveness of an ontology language like OWL. Hillairet et al. (2008) list some significant differences:

[4]http://www.omg.org/spec/MOF/2.4.2

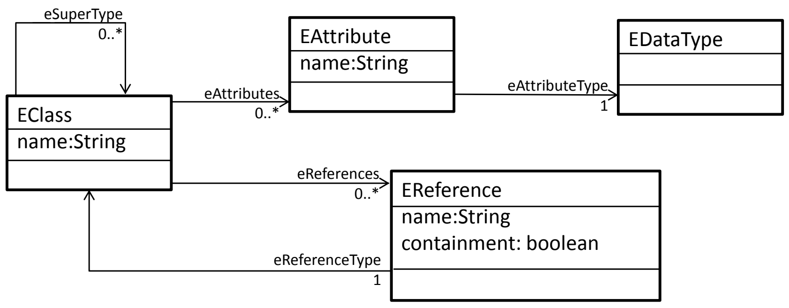


Figure 4.4 – Subset of the Ecore metamodel

- **Class membership:** In OWL an object can be a member of multiple classes. In EMF the membership is fixed to only one class.
- **Slot definition:** The properties in OWL are stand-alone entities that define their domain and range and can be used by any resource. EMF attributes and references are bound to their domain classes and can only be used there.
- **Structural inheritance:** In EMF, objects inherit their attributes from their parent classes. In OWL, properties are not inherited since they are not associated with a special class. Instead, properties define their own domains. These domains propagate upwards through the class hierarchy. In contrast to EMF, the language OWL supports a taxonomy for properties.
- **Flexibility**: OWL allows one to define new classes during runtime; in EMF this is not possible.

In Hillairet et al. (2008) some strategies are presented to overcome these restrictions. Another missing feature is an ontology reference layer for semantic reasoning on a set of asserted facts or axioms. This, e.g., can be based on first-order predicate logic and is relevant for semantic reasoning on world knowledge. Also for natural language understanding (NLU), the use of higher level inference mechanism is crucial in order to translate a text from natural language to a representation in an unambiguous formal language (Ovchinnikova, 2012).

However, in this thesis we work with the result of a natural language understanding component and the reasoning processes mostly focus on multimodal fusion, and discourse and context resolution. Related work showed that a language with an expressiveness that is compliant to TFS and additional reasoning algorithms like unification and overlay fulfill most of the requirements here. Since EMF expressiveness is compliant to TFS, it is suitable for use in SiAM-dp.

EMF is more than just a pure modelling framework. Models can be defined in three different representation forms and, outgoing from one form, the other two forms can be automatically generated. Figure 4.5 shows how EMF unifies three common modelling technologies: Java interfaces, XML Schema, and UML diagram. So it is possible to

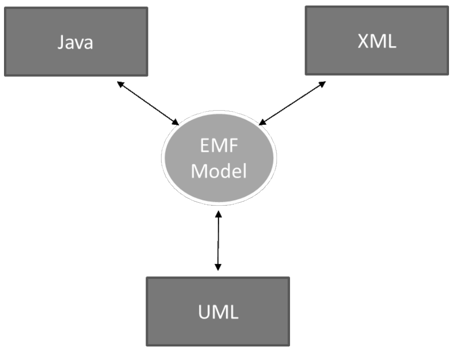


Figure 4.5 – Interrelation between EMF and Java, XML, and UML

define a model in the language of choice and generate EMF and the other languages from it. This implicitly means that XML serialisation of model instances is an integral part of the framework which is a required feature for the communication between the core dialogue platform and external components.

Furthermore, EMF allows one to automatically generate Java implementations (POJOs) from an Ecore model that can internally be used in the dialogue system for processing model instances. One of the biggest advantages of EMF is the integration into the Eclipse Workbench[5]. Outgoing from a model specification, a set of adapter classes can be generated that enable viewing and command-based editing of model instances in custom editors inside Eclipse. Besides that, a standard editor can be generated that directly allows the creation, access, manipulation, and validation of model instances in a tree or table editor view of Eclipse.

Finally, EMF contains a powerful framework for model persistence. It allows one to manage resources and to distribute them across documents or any other kind of storage. Furthermore, a highly customisable XML serialisation is supported. This makes it possible to directly integrate XML based language standards into our platform by representing them with Ecore models and customising their serialisation to the correct XML format. Some of the standards integrated into SiAM-dp are *Speech Recognition Grammar Specification (GRXML)*[6], *Speech Synthesis Markup Language (SSML)*[7], and *State Chart XML (SCXML).*

4.3.3 EMF API Extensions

Based on the Ecore meta-model we implemented an API that, among other functionalities, supports commonly used algorithms on typed feature structures that are very valuable for knowledge processing in multimodal dialogue systems. These algorithms

[5]http://www.eclipse.org/eclipse/
[6]http://www.w3.org/TR/speech-grammar/
[7]http://www.w3.org/TR/speech-synthesis/

are adopted from algorithms in the eTFS API as described in Pfleger (2007) and implemented for EMF models.

Unification

The API contains the static method `Unification.unify(EObject a, EObject b)` where `EObject` is the upper class for each Ecore model instance that - as discussed above - can be considered a typed feature structure. This commutative operation takes two typed features structures and returns a new typed feature structure that merges the information of both TFSs if they are compatible. Otherwise the result is *null*. Formally, Pfleger (2007) defines the unification as follows:

Definition 7 (Unification)
In an inheritance hierarchy of a typed feature structure the unification of two feature structures is their greatest lower bound (GLB). The GLB is the most specific type that subsumes the types of both TFSs.

A type subsumes another type if it is a superclass of it, e.g., the type *Person* subsumes the type *Man*. Figure 4.6 provides an example for a successful unification operation on two compatible typed feature structures.

A unification fails either if the two types of the TFSs do not subsume each other or if the values of a specific slot do not match. Figure 4.7 gives examples for both cases.

$$\text{unify(}\begin{bmatrix}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{genre:} & \text{Action} \\ \text{length:} & 130\end{bmatrix}, \begin{bmatrix}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{director:} & \begin{bmatrix}\textbf{Person} & \\ \text{name:} & \text{Shane Black}\end{bmatrix}\end{bmatrix}\text{)=}$$

$$\begin{bmatrix}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{director:} & \begin{bmatrix}\textbf{Person} & \\ \text{name:} & \text{Shane Black}\end{bmatrix} \\ \text{genre:} & \text{Action} \\ \text{length:} & 130\end{bmatrix}$$

Figure 4.6 – Example of the successful unification of two TFSs

$$\text{unify}(\begin{bmatrix}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{genre:} & \text{Action} \\ \text{length:} & 130\end{bmatrix}, \begin{bmatrix}\textbf{Person} & \\ \text{name:} & \text{Shane Black}\end{bmatrix}) = \text{null}$$

$$\text{unify}(\begin{bmatrix}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{genre:} & \text{Action}\end{bmatrix}, \begin{bmatrix}\textbf{Movie} & \\ \text{name:} & \text{The First Avenger} \\ \text{genre:} & \text{Action}\end{bmatrix}) = \text{null}$$

Figure 4.7 – Examples of failed unification. In the first example the types are not compatible. In the second one the TFSs have a conflicting value (*name*).

Strict Unification

The strict unification is a special implementation of the unification operation. The significant difference is the handling of multi-slot content. The standard unification approach does not consider the order of the entries in a multi-slot. Thus, two typed feature structures also unify if the content entities of a mulit-slot match but appear in a different order. In some situations, the order of multi-slot entries plays a relevant semantic role. In this case two typed feature structures should only unify if the order also matches. The static method `StrictUnification.unify(EObject a, EObject b)` provides an algorithm for this.

Overlay

The overlay operation is a non-commutative operation on typed feature structures that was introduced by Alexandersson and Becker (Alexandersson and Becker, 2001, 2003; Alexandersson et al., 2006; Alexandersson and Becker, 2007). The basis for this operation is the unification on two TFS instances where one is called *covering* (co) and the other *background* (bg). In contrast to standard unification, the overlay operation never fails. If possible, the content from the covering is extended with additional content from the background. In the case of a conflict, at least the content of the covering is returned with the risk that some of the background information is possibly lost. The overlay operation is composed of two basic steps:

Assimilation: A direct subsumption relation is necessary between the covering and background object in order to merge them. In the case of a type-clash the background must be refined to the most specific supertype of both objects.

Overlay The overlay of content is similar to the unification operation with the difference that in the case of conflicts, the information from the background is overwritten by the content of the covering so that an overlay is always possible.

Figure 4.8 depicts an example of an overlay operation. In this example the movie name causes a conflict so that the name of the background entry is overwritten with the name of the covering. The other content is merged as known from standard unification.

Besides the effect that overlay never fails, the key-feature is a scoring mechanism for the resulting TFS that shows how well the two TFSs fit together. Two different types of score mechanism exist: one that reflects the structural consistency (Pfleger et al., 2002) and one that estimates the information distance between the two arguments (Alexandersson et al., 2004). This score, especially, plays a relevant role in discourse resolution with ambiguous content since it reflects the quality of the merge result. The static method `Overlay.overlay(EObject co, EObject bg)` provides an algorithm for the overlay operation.

$$unify(co, bg) = unify(\left[\begin{array}{ll}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{length:} & 130\end{array}\right], \left[\begin{array}{ll}\textbf{Movie} & \\ \text{name:} & \text{The First Avenger} \\ \text{genre:} & \text{Action}\end{array}\right]) = \left[\begin{array}{ll}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{genre:} & \text{Action} \\ \text{length:} & 130\end{array}\right]$$

Figure 4.8 – Example of an overlay operation

Restricted Unification

The restricted unification is a special case of unification that was introduced by Pfleger (2007) for pattern matching. In contrast to the standard unification it is a non-commutative operation. It is implemented in the static method `RestrictedUnification.runify( EObject a, EObject b)`. The main difference to standard unification is that the first argument defines a pattern that at least must be comprised by the second argument. Thus, our unification example would fail with restricted unification (see Figure 4.9).

Although the algorithm for restricted unification played a relevant role in the predecessor dialogue platform ODP, it is supported only for the sake of completeness. Instead, SiAM-dp introduces the more powerful *PPattern* model (see 4.4) for pattern matching.

$$\text{unify}(\left[\begin{array}{ll}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{genre:} & \text{Action} \\ \text{length:} & 130\end{array}\right], \left[\begin{array}{ll}\textbf{Movie} & \\ \text{name:} & \text{Iron Man 3} \\ \text{director:} & \left[\begin{array}{ll}\textbf{Person} & \\ \text{name:} & \text{Shane Black}\end{array}\right]\end{array}\right]) = \text{null}$$

Figure 4.9 – Example of a failed restricted unification

Clone

Because the components of the dialogue platform often work independently from each other it is necessary to make deep copies of instances. Thus, it is possible to change the content of a copied instance without affecting the original one. With the method `EcoreUtil.copy( EObject eObject)`, the EMF framework already provides such a functionality, unfortunately with an unwanted effect:

EMF provides two types of references: *containment* and *non-containment*. A containment reference refers to an object that belongs to the object which defines the reference, a non-containment reference refers to an object that is content of another element or is a root element. In other words, you can say a non-containment reference is a plain *"A knows B"* relation, a containment reference is a *"A has B"* relation.

The EMF copy-method actually creates copies of objects that are the content of containment references and even considers non-containment references and redirects these references to the copied objects (compare Figure 4.10). The figure shows that *Entity_0* and the objects in its containment references (solid lines) *Entity_1* and *Entity_2* are copied. The non-containment reference (dashed line) from *Entity_3* to *Entity_2* is redirected from *Entity_6* to the newly created *Entity_5*. Nevertheless, if a non-containment reference refers to an object not contained in the actual instance, no copy is created and the reference is still directed to the original entity (*Entity_1*).

Since this behaviour is unwanted, we provide the method `EmfUtils.clone( EObject eObject)` with a full deep-copy functionality that also copies the external content of non-containment references (*Entity_10*).

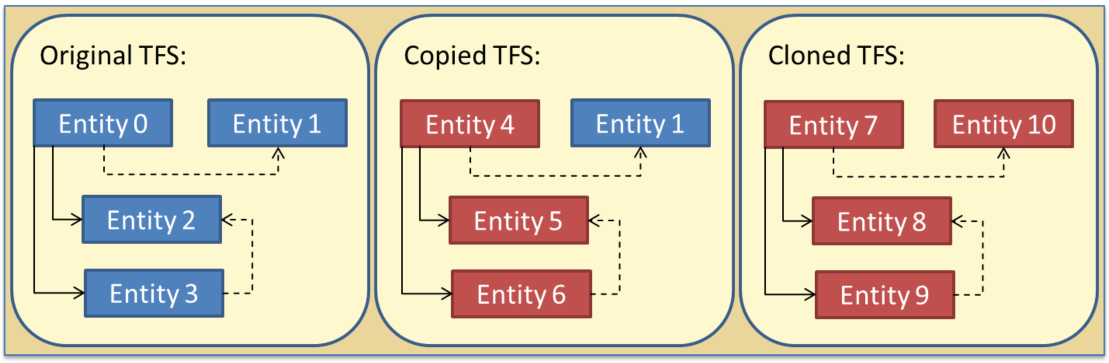


Figure 4.10 – Comparison of the results from the copy and clone method. The red marked entities are newly created during the copy or clone process.

4.3.4 Declaration of dynamic content: The Bindable Concept

The development approach of SiAM-dp supports the declarative specification of dialogue applications with EMF models. With the standard modelling solution, this has the big disadvantage that all contents of a model instance must be known already at time of design only allowing one to specify static information. With the newly introduced *Bindable*-Concept, we overcome this handicap and support instances whose contents can dynamically and context dependently be estimated during runtime.

The main idea behind the bindable concept is that objects or instances can have two different states (or simultaneously both for bindable objects). The first one is the static state, where the content of the instance is tightly predefined. This is the standard case when defining an instance, e.g., with the model editor. In the second state, the content of the instance is defined by an expression that is formulated in Apache Commons JEXL (Java Expression Language). This expression is evaluated just-in-time when the object is processed. We can say the content is *'bound'* to the evaluation result.

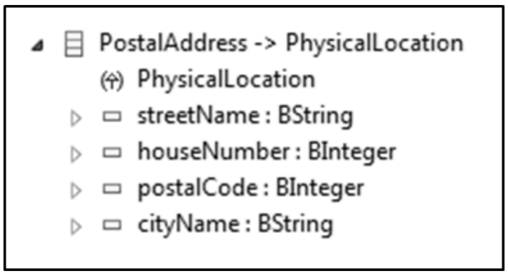
JEXL[8] is a library that implements an expression language supporting most of the constructs seen in ECMAScript which is a standardised scripting language and base of Java Script. The language supports all relevant literals, operators, and conditionals, thus providing an extensive scripting syntax. Furthermore, two variable types are available: Local Variables are valid in the actual script scope and declared by the syntax `var x`. Context Variables are defined and valid in the scope of a specific context. This context is provided by the model containing the JEXL-expression. E.g., the context of a dialogue model is the variable scope of the active node in a flow or statechart (see Section 6.1). Since the JEXL-API is written in Java, it is also possible to register Java objects as plugins and perform method calls on these objects directly from the script.

We differentiate between two types of *bindables*:

Bindable Datatype
Bindable Datatypes are the bindable versions of atomic datatypes. For every atomic datatype in Java we introduce a new bindable datatype that is derived from the abstract common class `BDataType`. For example, a `BString` extends the String, a `BFloat` the Float, `BInteger` the Integer datatype, and so on. Normally a datatype can only have a fixed value of this specific datatype. The bindable datatype can additionally be defined by a script expression. To distinguish between a fixed value and a bound value, the keyword `$expr(...)` is used. When the application developer declaratively fills the content of a datatype slot with a static value, he enters the string representation of the data which is then parsed using the standard `valueOf(String input)`-method of the datatype. If he wants to specify dynamic content, he uses the `$expr(...)`-keyword and specifies the script code inside the brackets. When the entity is processed, the script is evaluated with respect to the actual context and the result set as the content of the slot.

Figure 4.11a depicts the example concept `PostalAddress`, whose attributes are of the type `BString` and `BInteger`. Figure 4.11b shows an example instance of

[8]http://commons.apache.org/proper/commons-jexl/



PostalAddress -> PhysicalLocation
PhysicalLocation
streetName : BString
houseNumber : BInteger
postalCode : BInteger
cityName : BString

(a) The concept *PostalAddress*

PostalAddress	
cityName	“Cologne”
houseNumber	“23”
postalCode	“50667”
streetName	“Aachener Str.”

(b) The content of variable *neighbourAddress*

PostalAddress	
cityName	“Cologne”
houseNumber	“$expr(neighbourAddress.getHouseNumber() + 2)”
postalCode	“$expr(neighbourAddress.getPostalCode())”
streetName	“Aachener Str.”

PostalAddress	
cityName	“Cologne”
houseNumber	“25”
postalCode	“50667”
streetName	“Aachener Str.”

(c) Instance of PostalAddress with bindable datatype before and after script evaluation

Figure 4.11 – Example of an instance with bindable datatype content

this concept that is assigned to the variable `neighbourAddress` in the application context. This example only contains static values. Figure 4.11c shows an example of `PostalAddress` as possibly defined by an application developer. In the example the slots `cityName` and `streetName` are filled with static content. The attributes `houseNumber` and `postalCode` contain dynamic values described with JEXL-expressions which refer to the content of the variable `neighbourAddress`. During runtime the slots are filled with the results of the script evaluation as demonstrated in the figure.

Bindable Object

Bindable objects are the bindable versions of complex data types. They must be derived from the common abstract class `BObject` and inherit the attribute `binding` from this class. `BObject` is part of the SiAM-dp base ontology that also contains the concept `Entity` which is a child concept of `BObject`. Every semantic entity concept should be derived from this `Entity` concept and thus is automatically a bindable object. Furthermore, every concept of the SiAM-dp model is derived from BObject and therefore bindable.

In contrast to the bindable datatype, a bindable object is hybrid and can simultaneously contain static and dynamic content. The static content is defined by the developer during development time. The dynamic content is specified by a JEXL-expression in the `binding` property. This expression is evaluated during runtime and the result merged with the static content. Merging is realised by the overlay operation (see 4.3.3) at which the dynamic content constitutes the background and the static content the covering. Thus, in the case of a conflict, the information from the static content overwrites the information from the dynamic content.

Figure 4.12 shows the example of an object which specifies a binding. The instance of type Person contains a postal address that is bound to the variable `neigbourAddress` as defined in Figure 4.11b. During runtime, the content is replaced by the content of this variable. Only the nonmatching content for `cityName` is kept by the overlay operation.

$$
\begin{bmatrix}
\textbf{Person} & \\
\text{firstname} & \text{“Thomas”} \\
\text{lastname} & \text{“Mueller”} \\
\text{address} & \begin{bmatrix} \textbf{PostalAddress} & \\ \text{binding} & \text{“neighbourAddress”} \\ \text{cityName} & \text{”Köln”} \end{bmatrix}
\end{bmatrix}
$$

$$\Downarrow$$

$$
\begin{bmatrix}
\textbf{Person} & \\
\text{firstname} & \text{“Thomas”} \\
\text{lastname} & \text{“Mueller”} \\
\text{address} & \begin{bmatrix} \textbf{PostalAddress} & \\ \text{cityName} & \text{“Köln”} \\ \text{houseNumber} & \text{“23”} \\ \text{postalCode} & \text{“50667”} \\ \text{streetName} & \text{“Aachener Str.”} \end{bmatrix}
\end{bmatrix}
$$

Figure 4.12 – Example for the evaluation of a bindable object

4.4 Pattern Model

The predecessor dialogue platform ODP used restricted unification (see Section 4.3.3) for pattern matching on typed feature structures. This mechanism was excessively used in the matching phase of the integrated rule-based system. Nevertheless, for the use in SiAM-dp, we identified some disadvantages leading us to develop our own pattern model in order to overcome the following restrictions:

- Restricted unification works on TFS instances. When formulating a pattern based on abstract concepts, we face the problem that EMF does not permit one to create instances from abstract concepts. Thus, we would be forced to model type hierarchies without the use of abstract concepts which would limit our modelling potential.
- Restricted unification only supports the exists-quantor($\exists$). We also want to support the not-exists-quantor($\nexists$) and for-all-quantor($\forall$). The latter is especially suitable for multi-slots.
- The value restriction on atomic datatypes only supports the equals-operator for validation. In our use cases it turned out that additional operators like relational operators, logical operators (AND, OR), or even arbitrary script expressions are required.

Taking into account the above-mentioned requirements, a model for pattern definition was developed. In the dialogue platform it is used for different purposes.

Message Subscription:
Platform components can subscribe to messages distributed by the central event management. With the pattern model it is possible to restrict the subscription to only those messages whose content match a specific pattern.

Event Matching in dialogue specification:
The dialogue model (see Section 6.1) allows one to set a condition to an event that triggers a transition in the internally used statechart engine. This condition can be defined by pattern model instances which are matched with incoming messages.

Describing unresolved content:
Unresolved content in the reference model (see Section 5.5.1) is more closely defined with the help of the pattern model.

Definition of model mapping rules:
SiAM-dp contains an internal rule-based engine for mapping syntactic content on semantic content and vice versa (see Section 5.3). The conditions for the rules are defined with the pattern model.

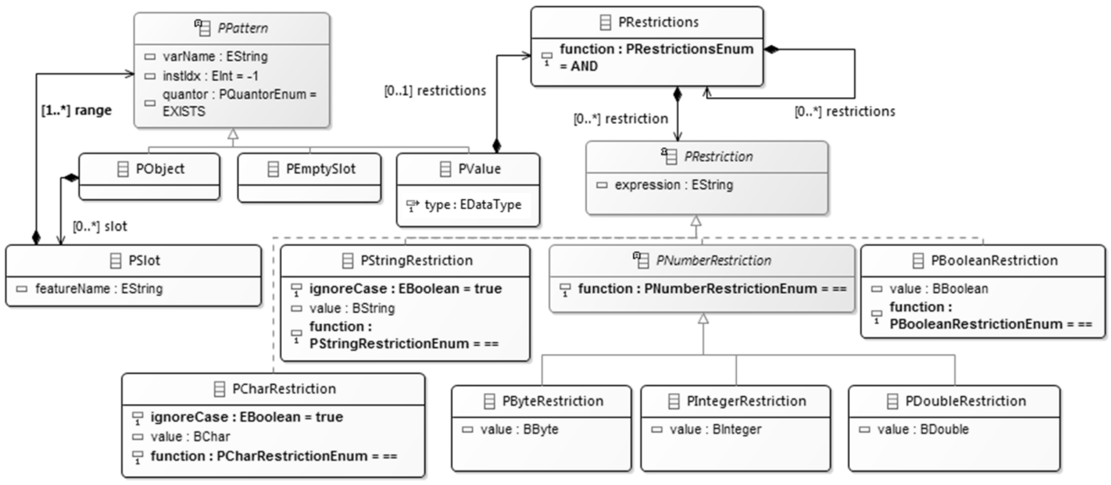


Figure 4.13 – The main concepts of the pattern model

4.4.1 Pattern Model Concepts

A first version of the pattern model was developed by Denerz (2013). This model has been adopted and extended for the requirements of SiAM-dp. Figure 4.13 shows the main concepts of this pattern model which are outlined in this section.

PPattern

`PPattern` is the abstract parent concept of all patterns. A pattern can specify the root entity or the content of a slot. It contains the following attributes:

`quantor` - this attribute sets the quantor for the pattern. Possible values are exists, not-exists, and for-all ($\exists, \nexists, \forall$).

`varName` - the pattern matching algorithm allows one to assign matching content to variables. Thus, the matching content can easily be accessed from the result of the match report. This attribute can only be used in combination with the exists-quantor.

`instIndx` - in the case of multi-slot features, this attribute can demand an exact index position for the content. Otherwise, the position is not considered. This attribute can only be used in combination with the exists-quantor.

PObject

`PObject` defines patterns for model instances. An instance satisfies a PObject if its type is a subtype of the type in `PObject` and all `PSlots` of the `PObject` are satisfied. `PObject` contains the following attributes:

> `type` - defines the type of the instance.
>
> `slot` - contains any number of instances of `PSlot` that define restrictions on the content.

PSlot

`PSlot` makes restrictions on the slot content of a `PObject`. `PSlot` has the following attributes:

> `feature` - the feature specifies the slot of the containing `PObject` on which the restrictions are made.
>
> `range` - is of type `PPattern` and specifies a pattern that must match the content of the given feature. Depending on the feature's type, the `PSlot` may have any number of instances of `PObjects` or `PValues`. If the feature is a non-many feature, the slot is a normal slot with only one content element, otherwise it is a multi-slot.

For the exists-quantor, a `PSlot` with a *non-many* feature is satisfied by an instance if the range `PPattern` is satisfied by the subinstance or value in the instance at the `PSlot`'s feature. `PSlots` that are multi-slots are satisfied if there is a subset of subinstances or values in the instance at `PSlots`' feature, such that all of them are satisfied by one `PPattern` pairwise exclusively. That means for each `PObject` or `PValue`, there must be a subinstance or value which satisfies the PPattern and each sub instance or value may be used at maximum once. If the `instIndx`-attribute of a pattern is set, the pattern must match the multi-slot entry at the given index position. If the non-exists quantor is set, the specified pattern must not match any instance in the slot. If the for-all quantor is set, every instance must match this pattern.

PValue

`PValue`s define fillers for attribute slots. They can impose `PRestrictions` as value constraints. A `PValue` is satisfied by a value if the attached `PRestrictions` are satisfied or if no `PRestrictions` is set. `PValue` contains the following attributes:

> `type` - defines the datatype of the value.
>
> `restrictions` - may impose restrictions on the value.

PEmptySlot

A `PEmptySlot` claims that the content of a slot must be empty.

PRestrictions

With `PRestrictions`, conjunctions or disjunctions of value constraints are represented. Restrictions can be nested and contain the following attributes:

`function` - defines the logical operator for the collection of restrictions. `PRestrictions` with a function set to AND is satisfied by a value if all sub-restrictions are satisfied. If the function is set to OR, only one of the containing `PRestrictions` or `PRestriction` has to be satisfied.

`restriction` - contains an instance of type `PRestriction` that restricts the value of a slot.

`restrictions` - defines nested `PRestrictions`.

PRestriction

`PRestriction` is an abstract class. For every datatype the model contains a concrete concept, e.g., `PStringRestriction` or `PIntegerRestriction`. For example, a `PBooleanRestriction` can be used to specify constraints on values of the type `boolean`. `PRestriction` contains the following attributes:

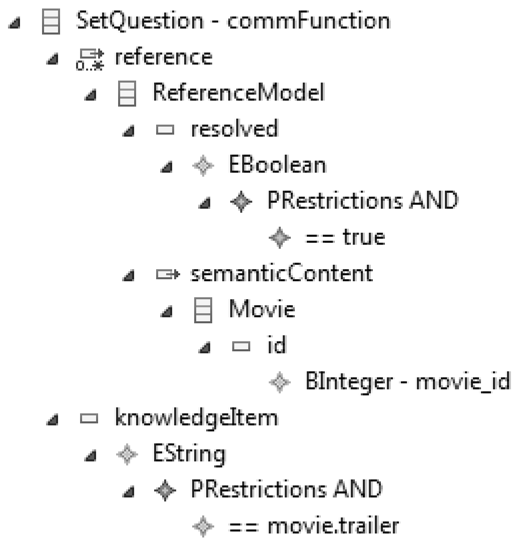
`value` - the value against which the content is validated.

`expression` - a JEXL expression whose result replaces the content of the value attribute during runtime.

`function` - the function that is used for validation. The type of the supported function is dependent on the concrete datatype. For example, the PBooleanRestriction supports the two functions EQUALS and NOT EQUALS. `PStringRestriction` additionally provides STARTS_WITH, ENDS_WITH, CONTAINS, or MATCHES for regular expressions.

4.4.2 Pattern Matching Example

Figure 4.14 shows a pattern example and a matching instance beneath. The type of the root object is set to `SetQuestion` and the result of a matching instance is assigned to the variable `commFunction`. The content of the multi-slot feature `reference` must contain one element of the type `ReferenceModel` whose feature `resolved` is a boolean value and restricted to the content `true`. Additionally, the `ReferenceModel` must contain one `semanticContent` of the type `Movie`. The `id` of the movie must be set but is not restricted to a value. The content of it should be assigned to the variable `movie_id`. Furthermore, the feature `knowledgeItem` of the `SetQuestion` must equal the string "movie.trailer". The given instance matches all these restrictions. After the match process, the complete `SetQuestion`-instance is assigned to the variable `commFunction`. The `movie_id` variable will be set to the value "13".



```
SetQuestion - commFunction
  reference
    ReferenceModel
      resolved
        EBoolean
          PRestrictions AND
            == true
      semanticContent
        Movie
          id
            BInteger - movie_id
  knowledgeItem
    EString
      PRestrictions AND
        == movie.trailer
```

$$\begin{bmatrix} \textbf{SetQuestion} & \\ \text{reference} & \begin{bmatrix} \textbf{ReferenceModel} & \\ \text{resolved} & \text{true} \\ \text{semanticContent} & \begin{bmatrix} \textbf{Movie} & \\ \text{id} & \text{“13”} \\ \text{name} & \text{”IronMan3”} \end{bmatrix} \end{bmatrix} \\ \text{knowldegeItem} & \text{“movie.trailer”} \end{bmatrix}$$

Figure 4.14 – A pattern example and a matching instance for this pattern

4.5 Summary

This chapter dealt with the first research question:

1. ***Modelling Language:*** *Which requirements must be fulfilled by a meta-modelling language that is used for the declarative development of multimodal dialogue applications?*

First, the chapter described approaches for the semantic representation of knowledge and briefly discussed their expressiveness and qualification for use in a multimodal dialogue system. In a requirement analysis, we derived a set of features for the meta-modelling language which is used in SiAM.

In the next part the finally used Eclipse Modelling Framework (EMF) was introduced and it is argued why this framework fulfills the previously mentioned requirements. Additionally, several API extensions and their implementations are described. This comprises several algorithms for unification and overlay that turned out to be very valuable for knowledge processing in multimodal dialogue systems. Furthermore, the framework needed an additional functionality for cloning model instances.

The standard modelling solution only allows one to specify static content during design time. To overcome this restriction, the following section introduced the bindable concept which allows developers to define instances whose content is dynamically evaluated from script expressions during runtime, taking into account the current context.

Pattern matching is used throughout the dialogue platform in order to define semantic constraints on instances. The final part of the chapter introduced a pattern model which allows one to define patterns for EMF instances. Pattern matching in SiAM-dp respects type-hierarchies, multi-slots and can specify arbitrary functions that are used for the validation of content.

5 Massive Multimodality in Cyber-Physical Environments

5.1 Introduction

We introduced the term Cyber-physical Environment (CPE) in Section 2.4. One feature of a CPE is the high number of devices that are spread throughout the surrounding environment but are part of an extensive network in the Internet of Things. From the dialogue platform's point of view, all devices in the environment are possible input and output devices for the realisation of the interaction between users and the environment. Input devices can be either **controllers** that serve the user as devices for multimodal input or **sensors** that allow to recognise non-intrusive activities in the environment. Output devices can be **renderers** that are part of a multimodal output representation or **actuators** that are directly controlled by the dialogue platform as part of the intelligent environment.

In the CPE a multimodal dialogue system must be able to handle a great number of devices and modalities. A human in the environment is not bound to one specific computer or control interface anymore. In fact, he interacts inside the environment, he changes his position in the environment, and may switch the applied control interfaces in order to conduct changes to the environment. Thus, in the perception of the human, the interaction with the various interfaces blurs into an interaction with the environment independent from the currently involved devices of the CPE.

From this new interaction paradigm, new requirements for the usability of the system arise. From the perspective of the user, the heterogeneous set of devices and modalities must be smoothly integrated, even if the user moves inside the environment. Different situations or users may pose diverse demands to the applied devices and modalities. Thus, a free choice of modality and the arbitrary combination of modalities are imperative for the dialogue platform. The following requirements must be considered:

- Free choice of modalities and multimodal combinations

- Dynamic reconfiguration of the set of available devices
- Adaptation to device failure or unavailability
- High heterogeneity of input and output devices
- Modality independent representation of user and system intentions

These requirements are not fulfilled by the multimodal dialogue systems presented in the related work in Chapter 3. Here at most three modalities are integrated into one system and the set of modalities is fixed. Also, current commercial systems like Apple's Siri or modern in-car systems concentrate on two modalities, mostly GUI and speech, sometimes also gestures. This makes a major difference in the requirements of multimodal platforms in heavily instrumented environments that try to capture all human senses. The platform must be flexible enough to integrate a massive heterogeneous set of devices and modalities concurrently that can dynamically change during runtime. In addition, a great variety of actuators and sensors must be considered. We use the term **massively multimodality** to capture the extreme variety in input and output modalities.

The challenge to a massively multimodal system on a technical level is to support a uniform interface for devices that encapsulate the heterogeneity in protocols and technologies. For a better understanding of the heterogeneity, we first classify possible devices into several categories (Section 5.2). For the communication between devices and the platform, a common language is required that is introduced in Sections 5.3 and 5.4.

The following sections deal with the semantic representation of the meaning behind an interaction (Section 5.5). An important task here is the conversion from a pure syntactic representation of input and output to a semantic level and back. This issue is discussed in Section 5.6. Here a generic rule-based approach is introduced that shifts content from the syntactic to semantic level or vice versa. This allows the fast and easy integration of arbitrary devices by the declaration of mapping rules.

5.2 Device Classification

In this section, we propose a new classification scheme for devices in massively multimodal interaction systems in CPEs. The goal is to find groups that describe properties of devices concerning the way that the devices are used and the role they play in the interaction between user and CPEs. The conclusions of the classification are intended to be integrated into the input and output model described later in this chapter.

One distinction can be made between devices that provide **input** to the system and devices that receive **output** from the system. A second distinction is made between devices that appear as **user interfaces** and **non-interactive networked devices** that are the basic components of the CPE. The main difference is that a user interface supports direct communication between users and the environment. This means that each

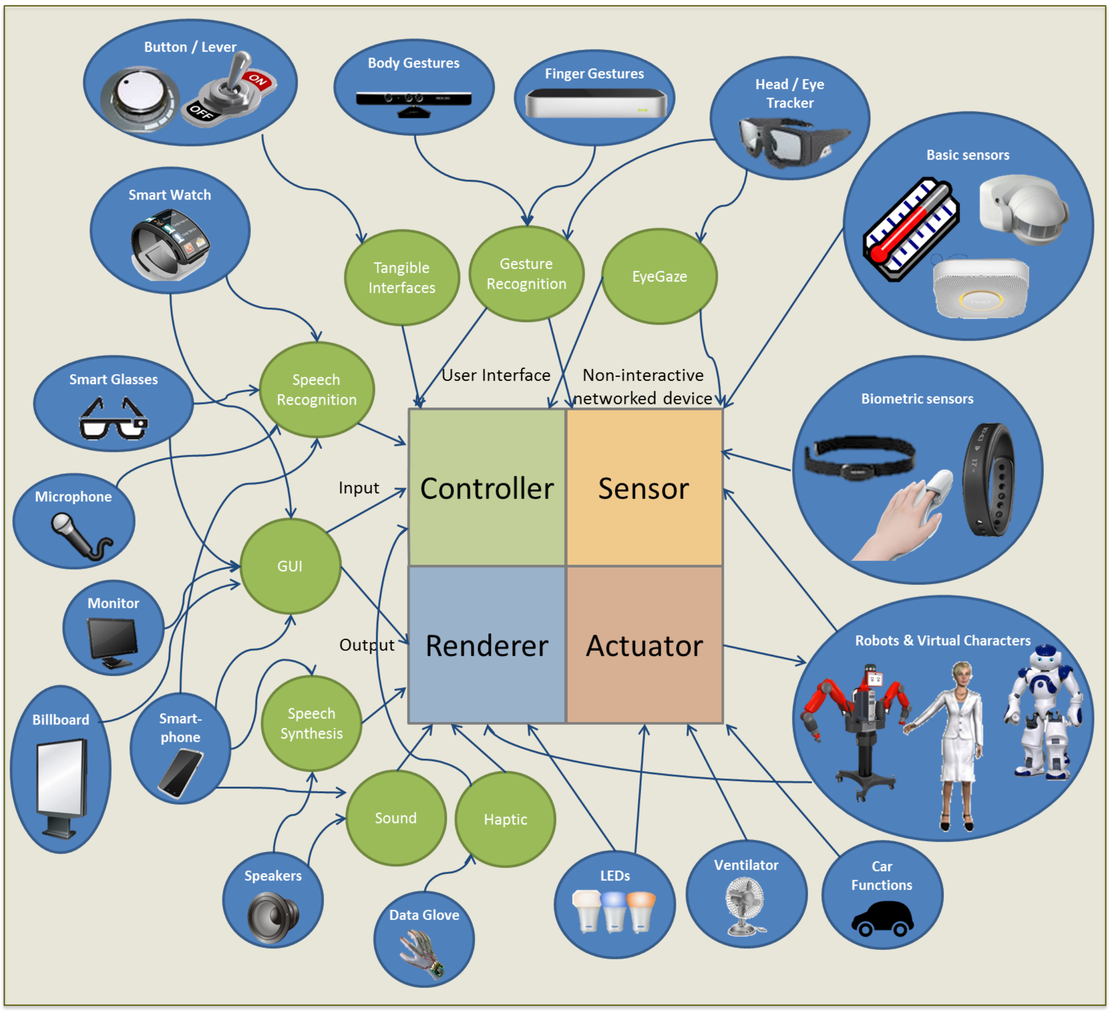


Figure 5.1 – The classification of devices in a massively multimodal system

interaction via a user interface transports a communicative meaning that is contributed to a dialogue between system and user.

The combination of both previously mentioned features results in four groups to which a device can be associated:

Controller - A controller is a user interface that provides input to the system. Thus, it constitutes a channel for the multimodal input of a user.

Sensor - A sensor is an input device that is used for non-intrusive activity recognition or the recognition of changes in the environment.

Actuator - An actuator is a static or movable device in the environment that can be controlled by the dialogue system in order to change the environment state.

Renderer - A renderer is an output device that presents information to the user. It is part of the multimodal output of the system.

Figure 5.1 shows the four classification categories and a set of devices that have already been used for the integration of use-cases with SiAM-dp (see Chapter 9). It additionally gives information about the modalities that are involved in the interaction between user and environment.

However, the classification of devices is not always unambiguous. In this case the devices are associated to more than one of the categories. For example, a smartphone already constitutes a multimodal interface with the support of a Graphical User Interface (GUI) and speech interaction in both communication directions. Furthermore, it is the carrier of a wealth of sensors that can actively or passively contribute to a dialogue application. The same applies to other smart devices like the smart watch or smart glasses. Other sensors like a body motion sensor can detect the users's intentional gestures but also recognise activities. A robot may communicate with a user by speech or perform physical actions like moving an arm or walking through the room.

Therefore, an advisable approach is not to classify the device as a whole but first to identify the separate services of the device that contribute to the dialogue system. The classification will then be made for the services. In Chapter 9, concrete applications are presented that are built with the SiAM-dp platform. They integrate many of the devices presented in Figure 5.1. For each application a table will show the supported devices and a classification of the applied device channels.

In the following we present some additional features that can be used for the classification of a device. These features are orthogonal to the previously introduced features. Figure 5.2 shows the classification of a selection of devices by these features.

Device Position:

Wearable - A device that is worn by the user under, with, or on top of clothing.

Immobile - A device with a static location in the environment.

Mobile - A device with a changeable location in the environment.

Interaction Range:

Near-field - The interaction takes place at a close distance to the device.

Far-field - Interaction is possible from an arbitrary position in the room.

Number of Users:

Single-user - Only one person uses the device.

Multi-user - Many persons use the device.

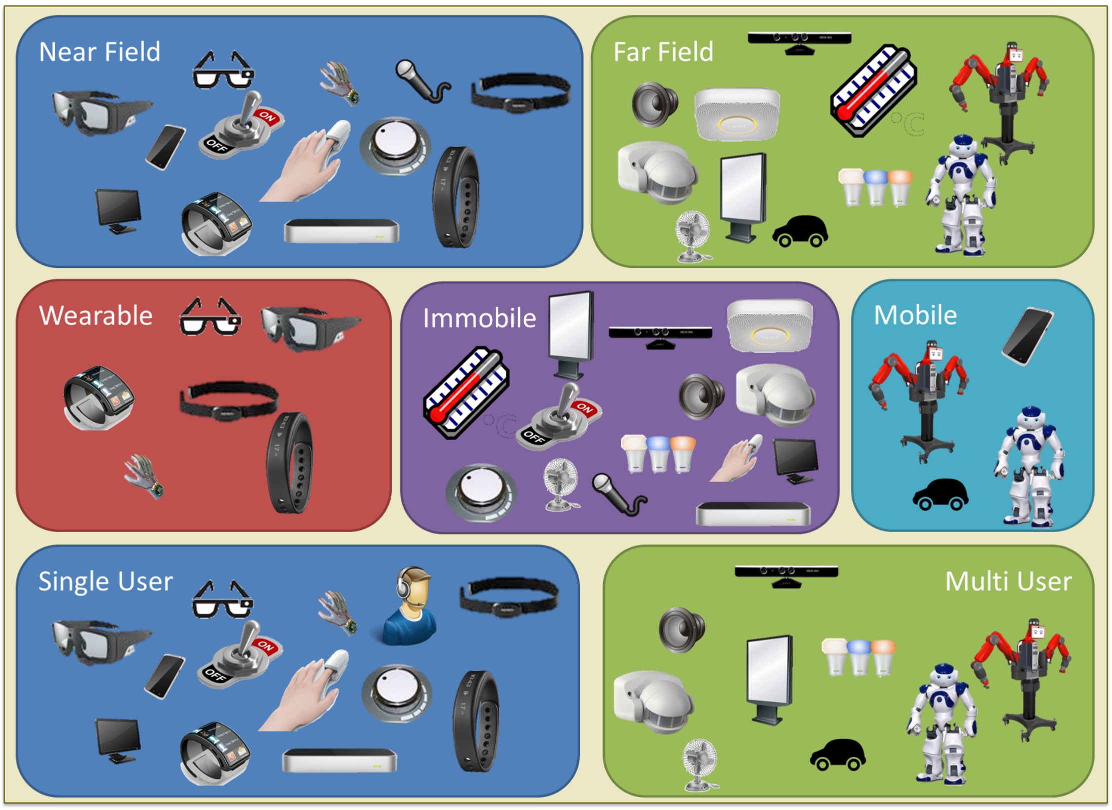


Figure 5.2 – The classification of devices in a massively multimodal system into features orthogonal to the device type

5.3 Representing Input and Output

One of our requirement specifications was that a heterogeneous set of input and output modalities can be integrated into the system. They differ in terms of the type of information they provide and the protocol they use for transmitting information to the dialogue system and vice versa. This creates the problem that the system in a first step is not able to handle all the various information in a well defined procedure due to a lack of common and structured representation.

Towards the goal to achieve a comprehensive processing of input and generation of output for all devices, modalities, and technologies, it is necessary to represent all input and output messages in a common model that is used throughout the complete dialogue platform and serves as an interface between the dialogue system and external modules with their proprietary representations. This model must be open and flexible enough to integrate all the various modalities and accordingly has to integrate modality specific content descriptions. This particularly means that concepts for important modalities such as speech input or graphical user interfaces should already be supported. For example, for speech input representation the model should contain predefined structures that contain further meta-information besides the recognised utterance, like recognition confidence, word lattices, or phonetic transcriptions that are provided by state-of-the-art speech recognisers by default. For the most common modalities the model should already propose an elaborated representation but the flexibility of the system demands that the model can be freely extended if new modalities are integrated. In order to maintain the compatibility to existing standards like, e.g., GRXML for speech recognition grammars or SSML for speech synthesis the specific structures should also allow one to incorporate external language specifications.

In Section 3.4 we discussed the advantages of separating dialogue acts into the representation of communicative behaviour and communicative function. The IO model should also encourage this idea by making information representable on two distinct levels of abstraction:

Syntactic level: Embraces annotations for common meta data of a dialogue act such as the begin time, duration of the action, device ID, modality, initiator, or addressee. This level also includes modality-specific content that describes the communicative behaviour.

Semantic level: The semantic representation is located on a more abstract level. Here the content is device independent and incorporates the communicative function of the dialogue act as well as the thereby transported semantic content.

In Section 3.3 we introduced some markup languages for modelling multimodal interaction. The most sophisticated and already standardised language is the Extensible MultiModal Annotation markup language (EMMA). After a systematic examination of the standard, we decided against the use of this standard and for the development of our own model for input and output representation for the following reasons:

1. EMMA has been primarily developed for the representation and annotation of user input. Our model should additionally can represent output messages that are distributed through a heterogeneous set of output modalities.
2. The model should enable the communication of sensor input information and control messages to actuators. This is also not supported by EMMA
3. EMMA is not modelled in EMF and thus does not allow easy combination with other models of SiAM-dp.
4. The representation of communicative functions is only based on strings that are not predefined by the standard. This does not apply to our approach to represent communicative acts with the standardised dialogue act model as presented in Section 5.5.
5. The EMMA model is not built on a type hierarchy. Thus, unification and pattern matching that are based on inheritance in the type hierarchies cannot be exploited.

Nevertheless, many ideas and concepts of EMMA have been incorporated into this work.

5.3.1 The IO-Model Type Hierarchy

While developing the model for input and output representation, we pursue two major objectives. First, the model should serve as an interface between external input and output devices and the core dialogue system. Second is to provide help structures that support situation-adaptive multimodal behaviour of the system.

On the input side it must be possible to consider the uncertainty of an input and to deliver several interpretation hypotheses that can be later evaluated by the system taking into account other input modalities or the current context. On the output side it should be possible to suggest distinct alternatives inside the model. Based on these alternatives the situation adaptive system selects the presentation that is most suitable to the current context, situation, and available devices.

Figure 5.3 gives an overview of the top-level type hierarchy of the IO-model. In the following sections we often use diagrams in order to depict the models in SiAM-dp. A box in this diagram describes a concept, the name of the concept is the header of the box. If the name is written in cursive letters, the concept is an abstract concept. The entries in the main part of the box define the names, types and cardinality of the concept's attributes. Inheritance between concepts is indicated by a line with a closed, unfilled arrowhead pointing at the superclass. If this line is dashed, the superclass is an abstract concept. Associations are indicated by a solid line, the direction of the association is expressed by an open arrowhead. In the following sections an association will also be

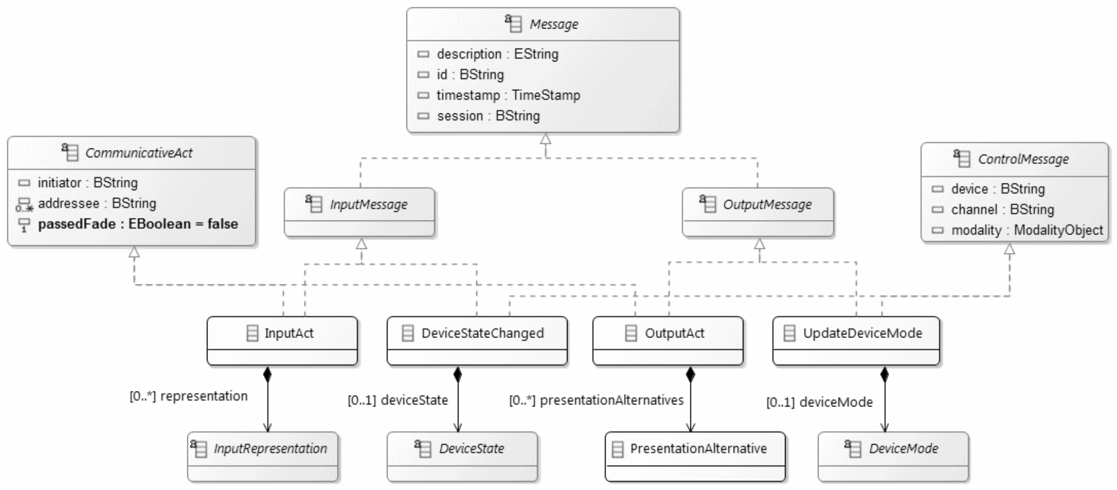


Figure 5.3 – The upper level of the IO-model type hierarchy

called a reference to another concept. The label of the line gives information about the cardinality and name of the reference.

In the IO-model every message that is exchanged between external and core components of the dialogue platform is represented with an instance of the abstract concept *Message* which is also the top concept of the type hierarchy. It defines some general attributes like a unique identifier for the message, a timestamp, and the current session, to which the message is associated.

The four main types of concrete message realisations can be classified according to two categories:

- The communication direction of the message is distinguished by the two abstract concepts *InputMessage* and *OutputMessage*. *InputMessage*s are messages sent from external components to the core system. The other way around, *OutputMessage*s are sent from the core system to external components.
- The second distinction is made between communicative acts and control messages that are described by the abstract concepts *CommunicativeAct* and *ControlMessage*. Communicative acts transport messages with a communicative intention between users, dialogue application, and the CPE. Control messages are used for device setup and configuration and can either update the state of a device from the dialogue application or report on state changes.

Combinations of these categories lead to the four concrete upper message concepts:

InputAct: Input from the user with a communicative intention.

OutputAct: Output to the user with a communicative intention.

DeviceStateChange: Notification about device state changes externally initiated.

UpdateDeviceMode: Device state change initiated by the dialogue system.

5.3.2 Communicative Acts

Subclasses of the *CommunicativeAct* inherit attributes that describe the *initiator* and *addressee* of an interaction. In the following, the derived concepts *InputAct* and *OutputAct* are introduced:

Input Acts

Figure 5.4 shows the diagram of the *InputAct* concept. Generally one can assume that a user input contains the syntactic representation of the input and hypotheses about its interpretation. Therefore, the *InputAct* comprises two content types:

InputRepresentation
The input representation embraces certain data of an input act. It contains common meta data, such as device identifier, modality, begin time, and duration of the interaction. From this the abstract classes *SensorInput*, *ControllerInput* are derived. These are the parent classes for all concrete device specific concepts for the representation of input (see Section 5.4). The IO-model already comprises concepts for common input devices but can arbitrarily be extended by new concepts. Furthermore, it is possible to represent an input act in diverse formats and granularities by appending more than one representation of an input. For example a pointing gesture on a touchscreen can be represented only with the associated coordinates of the pointing gesture or as a click event on an eventually displayed GUI element.

Hypothesis
The slot `hypotheses` contains possible interpretations of the input. Every hypothesis in this slot can be annotated with a confidence value of the interval [0..1] that reflects the recognition and interpretation certainty. Thus, more than one possible interpretation result with distinct certainty can be passed to the dialogue system. The decision about the effectively processed interpretation is made later, based on the dialogue or environment context. Figure 5.4 contains three device specific hypotheses. The *SpeechHypotheses* contains information about the outcome of a speech recogniser, the text of the recognised phrase. Respectively, the *GestureHypothesis* contains assumptions about a performed gesture and *PointingHypotheses* about the target of a pointing gesture or a gaze. Every hypothesis can contain a *CommunicativeFunction* instance. Here, the semantic result of the interpretation process is described with the device independent dialogue act model (Section 5.5).

Often in a standard interaction workflow, an *InputAct* instance is not directly filled by the input device on the whole. Moreover, the content is continuously added by the individual input processing modules.

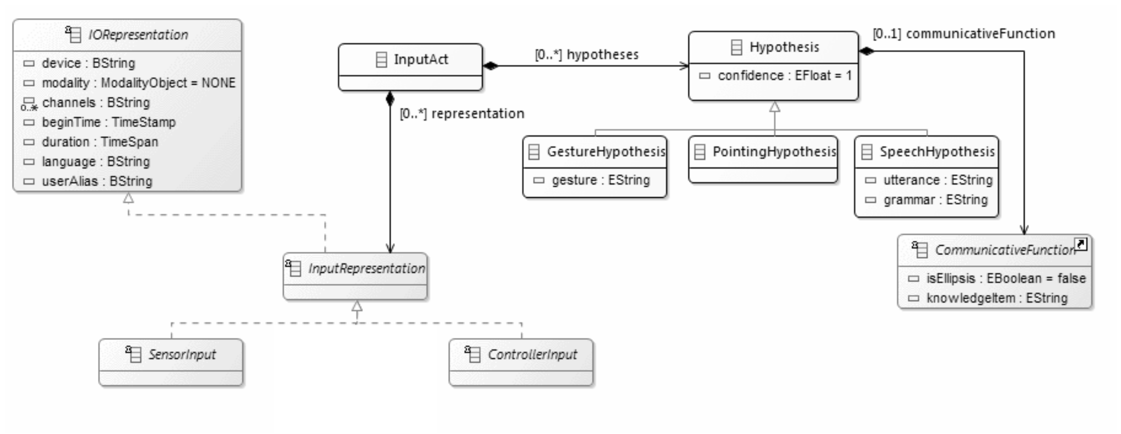


Figure 5.4 – Graphical representation of the EMF specification of the *InputAct* concept

Output Acts

The output of the system is generated by the dialogue management component. One significant difference to the input is that the output of the system and its communicative intention is not based on hypotheses, it is well defined. Thus, the semantic representation of the communicative function is a direct content of the *OutputAct*. Figure 5.5 shows a diagram of the *OutputAct*-concept.

SiAM-dp is a situation adaptive system which means that the actually used modalities and the mode of presentation can differ depending on the current context. For this the model has to allow content to be provided in multiple alternative representations, some usually being more specific than others. Hence, one *OutputAct* can contain one or more *PresentationAlternatives* that serve as the foundation for a situation-dependent behaviour. Based on the current context, available devices, and users, the system can evaluate and select the most suitable alternative and forward its containing *OutputRepresentations* to the affected output devices.

A *PresentationAlternative* can again consist of one or more *OutputRepresentations*, allowing the distribution of the system's output to several devices, e.g., a combination of speech output with a GUI presentation. The abstract concept *OutputRepresentation* is basis object for all modality specific output representations. It collects common attributes like the device ID, modality type, or begin time and duration of the output. For a better classification, the abstract concepts *RendererOutput* and *ActuatorOutput* are derived from this concept. Modality specific content is described in concrete concepts that are derived from them (see also Section 5.4). Our IO-model already provides concepts for the most common output devices. Nevertheless, the model is easily extendable

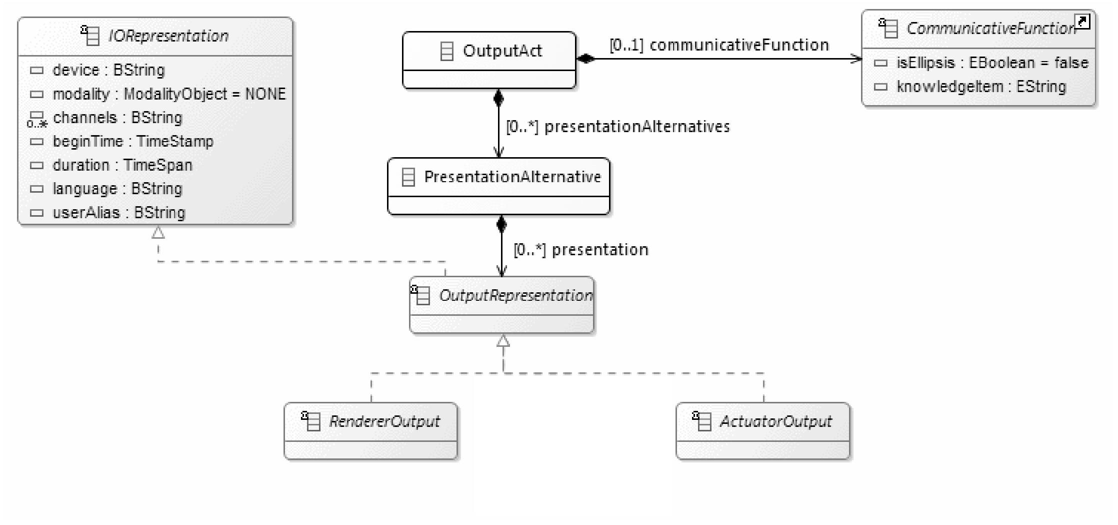


Figure 5.5 – Graphical representation of the EMF specification of the *OutputAct* concept

with new devices if necessary.

An output act is enriched with additional content while being processed by the particular interaction workflow components for presentation planning and multimodal fission.

5.3.3 Control Messages

Subclasses of the abstract concept `ControlMessage` inherit attributes that describe the device ID, channel, and modality of the affected device. Figure 5.6 shows a diagram of the derived subconcepts.

Update Device Modes

When the dialogue system triggers the change of a device mode or configuration, an instance of this concept is sent to the device. The instance contains the new device mode that should be adopted. Every description of a new device state must be derived from the abstract concept `DeviceMode`. Examples for such device modes include the request to cancel a presentation, update the grammar of a speech recogniser, or set the speech recognition mode to push-to-talk or speak-to-activate. Besides the already supported device modes, the model can be arbitrarily extended with modes for new or existing devices.

Device State Change

The change of a device state is propagated to the dialogue system with an instance of the `DeviceStateChanged` message. It contains an instance of the abstract concept `DeviceState` which describes the change. This can be information about a started or stopped presentation or about a started or rejected speech input. The states that are part of the base model can be extended with device states for arbitrary devices.

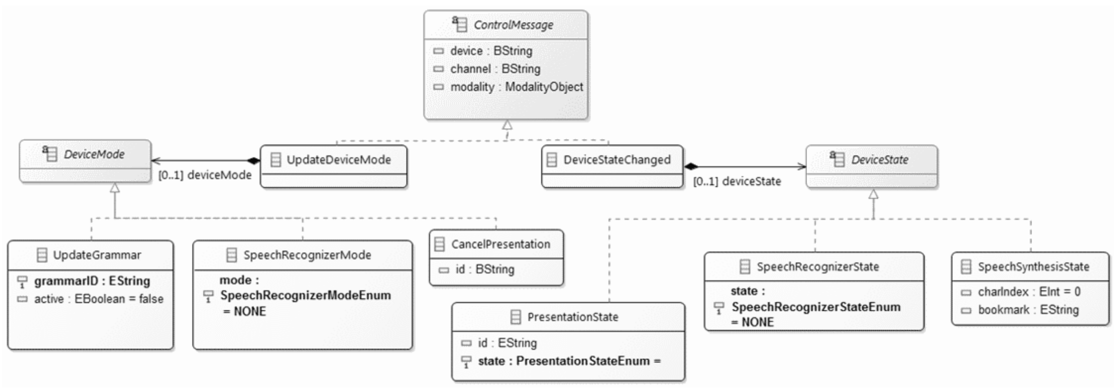


Figure 5.6 – Graphical representation of the EMF specification of the *ControlMessage* concept

5.4 Massively multimodal integration

In the previous sections we presented an approach for classifying devices in a CPE and the model that is used for the representation of input and output in a multimodal dialogue application. This model already contains entry points for the representation of device specific input and output. To sum up, these are the abstract concepts *SensorInput*, *ControllerInput*, *RendererOutput*, and *ActuatorOutput*. These concepts are directly derived from the main features for classification we identified in Section 5.2.

Figure 5.7 shows a snippet of the type hierarchy outgoing from these entry points. For every representation type it contains some examples for device specific representations as they have been used in various use-cases (see Chapter 9). The branch for the concept *Gesture* shows that within this tree it is possible to specify sub-hierarchies. Here, the concepts *HeadGesture*, *HandGesture* and *BodyGesture* are collected under the concept *Gesture*.

The other categories that have been identified as appropriate classification features in Section 5.2 can be realised as attributes for every representation. These attributes are defined in the root concept *IORepresentation*. In detail these are the attributes *modality*, *position*, *interactionRange*, and *userNumber*.

The base hierarchy already contains concepts for many common devices. Nevertheless, the world of CPEs is very heterogeneous, making it impossible to cover every device type. Therefore, it is intended that the model is domain specifically extendable with

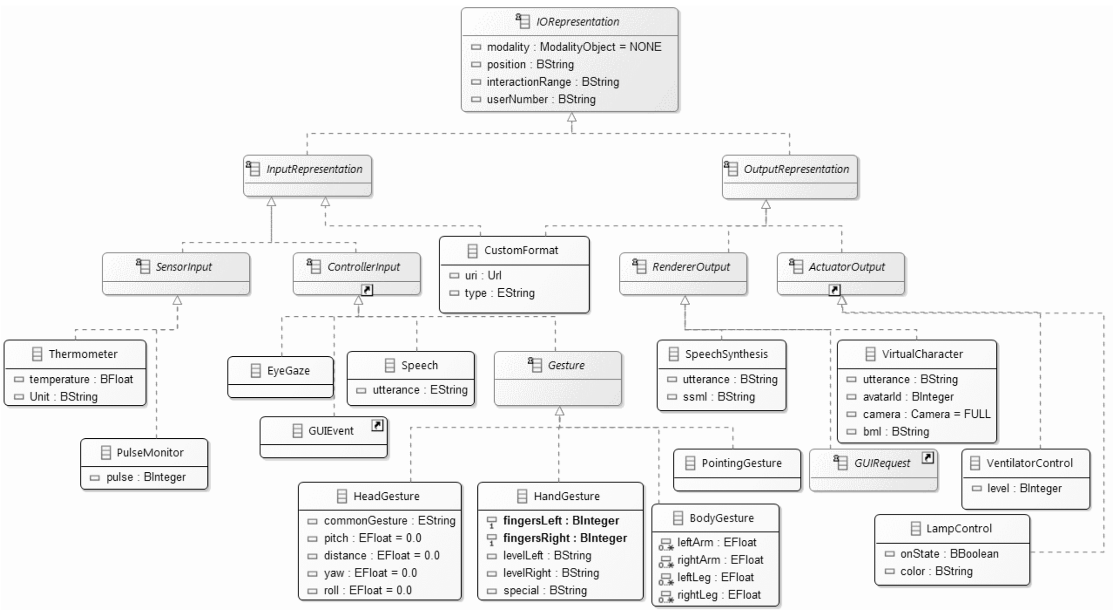


Figure 5.7 – Snippet of the type hierarchy for the representation of input and output events

representations for new devices by deriving more concrete concepts from concepts that are already part of the hierarchy tree. In the following we present how the representation is realised in practice for some selected devices.

Custom Format: The concept *CustomFormat* is used for the non-standard representation of data. This has to be accompanied by a format identifier and should only be processed or interpreted by components that support this particular format. The data is given with a list of key value pairs. Additionally, a URL can be provided from which the custom format data should be retrieved if desired (if it is considered too large to include in the message). Using the custom format, representations can quickly be provided without extending the underlying model. This is particularly useful for ad hoc solutions for prototyping or devices that do not fit into a specific category.

Graphical User Interfaces: Since SiAM-dp provides a specific concept for describing graphical user interfaces, the models for representing GUI output and the input events on GUIs are very well elaborated upon. The GUI model and related representations are described in detail in Section 6.3.

Speech Synthesis: The concept *SpeechSynthesis* for describing speech synthesis output contains two attributes. For the easy and fast synthesis of an utterance, the attribute *utterance* can be used. Furthermore, it is possible to define the output with the standard language for speech synthesis, the Speech Synthesis Markup Language (SSML) which is more powerful and e.g., allows one to define attributes like voice, prosody, break, or emphasis. The SSML syntax is given with the attribute *ssml*.

Speech Recognition: The concept *Speech* is used for the description of a speech input. Besides some meta-information, it provides hypotheses about the recognised utterance. Since this information is not certain, it is provided with the concept *SpeechHypothesis* in the *hypothesis* slot of the input act.

Tangibles: The abstract concept *Tangibles* is the root concept for tangible devices. Concepts that describe input from tangible devices should be derived from this concept.

Gestures and Postures: In the diagram the concept of gestures is divided into the three subconcepts *HeadGesture*, *BodyGesture* and *HandGesture*. The information provided by the three subconcepts differs in the content provided. While the hand gesture is described by coordinates of the fingers, a body posture may describe the complete skeleton of the body. A movement of the head can be described by the attributes pitch, yaw, and roll. Hypotheses about recognised gestures can be given in the hypothesis slot of the input act. Since the data from diverse devices can vary considerably, in most situations it is suitable to derive a new concrete concept from the *Gesture* tree.

Wearable devices: Most wearable devices can be considered a Cyber-physical

System (CPS) since they combine several sensors and interfaces. Thus, several representation formats are suitable for use with a wearable device. Smart glasses, smart watches as well as smart phones, are equipped with GUIs. Here the above-mentioned representation concepts for a GUI should be used. If speech interaction is supported, the models for speech are the right choice. Further sensors and buttons can be covered either by new concepts derived from the concepts *Sensors* or *Tangibles* or by the *CustomFormat*.

Virtual Characters: Similarly to the approach for speech synthesis virtual characters can be controlled in two ways. The first is to send API specific utterances that are directly transcribed by the connected engine for presenting virtual characters. The second is to use a standardised Agent BML (Behaviour Modelling Language) model which describes an interaction that a virtual character should perform. This may also include speech.

Actuators: A complete taxonomy can be developed outgoing from the abstract concept *ActuatorOutput*. In the diagram this is exemplarily done for the control of a multicolour lamp and a ventilator.

Sensors: A complete taxonomy can be developed outgoing from the abstract concept *SensorInput*. In the diagram, this is exemplarily done for a thermometer and a pulse monitor.

5.5 Semantic Dialogue Act Model

In Section 3.4 we discussed the fact that a disadvantage of representation on a purely syntactic level is that input or output messages from a device are very modality specific. As a result, every description of a dialogue workflow must be adapted to each specific device and modality. In practice, when specifying the dialogue workflow, the application developer has to explicitly react to the input of every input modality and send an output message for each output modality, respectively. In order to make the dialogue platform more flexible and adaptable to new modalities and devices, we introduce a model for dialogue acts that describes the user's or system's communicative intentions. Instances of this model contain the communicative intention of a dialogue act and eventually thereby provided semantic content. Thus, the model for controlling the dialogue workflow can be based on these semantic dialogue acts and be completely independent from the actually used modalities and devices.

The model for the definition of communicative functions is inspired by a standard for the semantic annotation of dialogue acts (ISO/DIS 24617) as introduced in Section 3.4.4. For the semantic description of dialogue acts, we adopted the type hierarchy of communicative acts that is specified by this standard and integrated it into our model for communicative functions (Figure 5.8 shows an excerpt of the upper level). This type

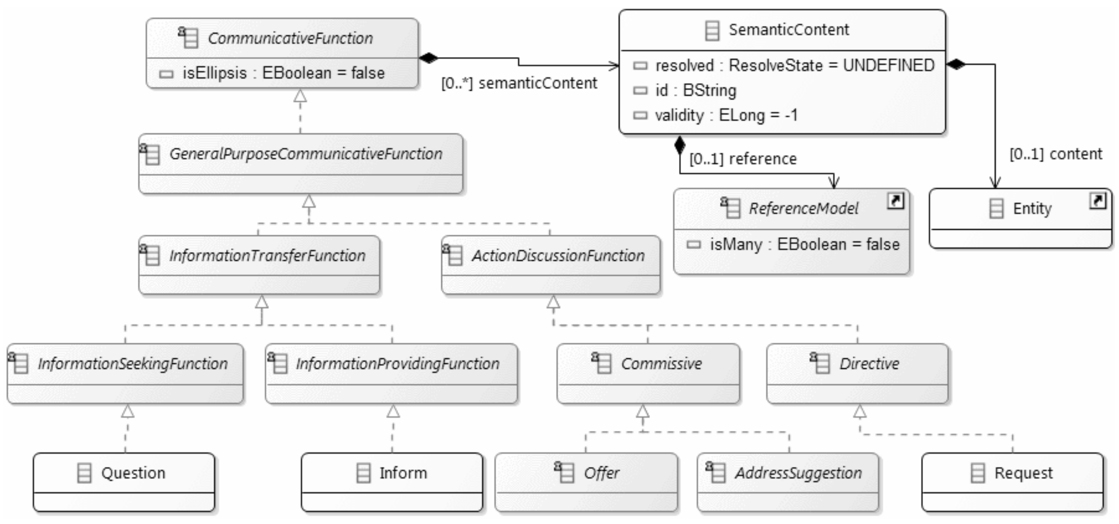


Figure 5.8 – Upper level excerpt of the communicative function type hierarchy

hierarchy, inter alia, contains concepts for seeking and providing information, offering and demanding tasks, or controlling the dialogue, e.g., turn taking or giving feedback.

The diagram shows that every communicative function can contain one or more elements of the type `SemanticContent`. This concept is a container for entities that are carried by the dialogue act. One can generally distinguish between resolved and unresolved semantic content elements and a `SemanticContent` instance can adopt one of the following two states:

resolved - Resolved semantic content is an entity that is introduced with the dialogue act into the discourse context. For example the utterance *"What is Iron Man about?"* introduces the movie *Iron Man* into the discourse.

unresolved - Unresolved semantic content contains referring expressions to entities that have already been introduced by a previous interaction turn (anaphora), will be subsequently introduced into the discourse context (cataphora), or are part of the environment context. The dialogue act in the utterance *"Who is starring in this movie?"* for example refers to an entity of type `Movie` that is not implicitly given but is already part of the discourse context.

Entities in the platform must derive from the base concept `Entity`. This concept also serves as the anchor point for new domain-specific concepts, thus the concepts in new domain specifications should be derived from this basic concept.

Already resolved content is added to the `content` slot of the `SemanticContent` concept. If the content is unresolved, the information of the referring expression is represented by an instance of the `ReferenceModel` (see next subsection) that uses concepts similar to the approach of Pfleger (2007). Instances of this model are added to the slot `reference`.

During runtime it is the responsibility of the fusion and discourse resolution engine to resolve the referring expressions and fill the `content` slot with entities from the context.

5.5.1 Modelling Referring Expressions

The concept diagram for the ReferenceModel is depicted in Figure 5.9. As discussed in Section 2.3.2 communicative acts can contain different types of referring expressions. The reference model of SiAM-dp already supports a subset of them, each represented by a specific concept derived from the abstract concept `ReferenceModel`. This upper concept contains the slot `referencePattern` where semantic restrictions on the referenced entities can be specified. Since referring expressions mainly appear in verbal interaction, it is also possible to define syntactic restrictions in the slot `hasMorphoSyntacticDecomposition`. The linguistic properties that can be specified here are taken from the LingInfo ontology (see Buitelaar et al. (2006)) and allow one to describe case, gender, part-of-speech, and number of a linguistic expressions.

The model contains the following concrete concepts for referring expressions:

Deictic Reference - Represents a deictic expression, for example in the utterance *"Give me information about this movie"* that occurs together with a pointing gesture in a multimodal input. A deictic referring expression can also appear without a gesture. In this case, the expression is handled as an anaphoric reference to content that has been introduced into the discourse context in the previous discourse. In the example, the movie has perhaps been mentioned in a input and is now referred to.

Knowledge Base Reference - This concept describes an exophoric reference to an entity that is part of the environment context. It is used for retrieving knowledge from the world that has not been previously introduced into discourse. For example the utterance *"Turn off the red lamp"* can refer to a lamp in the environment that

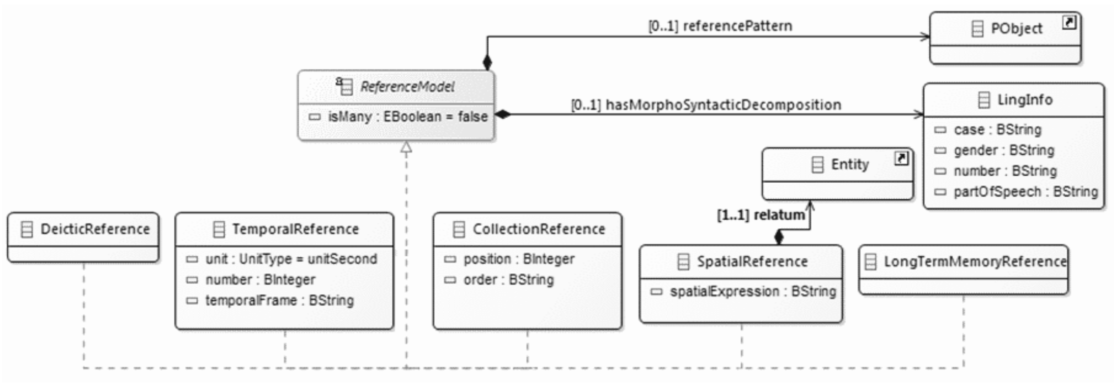


Figure 5.9 – The `ReferenceModel` class diagram

has not been previously mentioned in dialogue and thus is not part of the discourse context. Nevertheless, in an environment-aware system it is possible to resolve references to entities in the environment that are stored in the knowledge base.

Spatial Reference - A spatial expression is a reference to an object relative to the current location of the participant of an utterance or relative to other objects mentioned during the discourse. The SpatialReference object allows one to model the relatum of the referenced object and the spatial relation to it, e.g., with *northOf*, *leftOf*, *topOf*.

Collection Reference - Can express differentiation criteria in order to distinguish the referenced object out of a collection of possible referents, e.g., in a list of entries (*"Book a ticket for the second cinema!"*). The concept provides attributes for the starting point from which the resolution is applied and the ordinal number of the referred element.

Temporal Reference - An absolute temporal expression can specify the temporal frame of an interaction. This temporal frame is either the current time and date or a time point previously mentioned in a discourse. A relative temporal reference applies to this temporal frame. This reference can be quite precise (e.g., *"The movie next Monday."*) or specify a vague relation (e.g., *"I booked a later screening"*).

5.6 Mapping between syntactic and semantic representations

In the previous section we discussed the fact that the specification of the dialogue flow on a semantic level improves the expandability and maintainability of an application. For this it is necessary to perform a transformation from the syntactic to a semantic representation and vice versa during runtime. In SiAM-dp this task is done by specific components that are engaged in the processing workflow. On the input side interpreters lift the syntactic to a semantic representation. On the output side the generation of syntactic from a semantic representation is performed by renderers and actuator controllers (see also Section 7.1).

The core platform already integrates interpreters for speech input and input from GUIs. SiAM-dp deploys modelling languages for speech recognition grammars and GUI presentations. They allow application designers to directly annotate their models with annotations for the semantic interpretation of the users' inputs. Based on these annotations, the interpreters preprocess syntactically represented input and enhance it with a semantic representation. This procedure is explained in detail in Section 6.3 for GUIs and in Section 6.2 for speech input.

Similarly to this approach, it is possible to extend the platform with arbitrary interpreters and renderers. Mostly the approach here is to map values from one level of representation to the other one. Since this procedure can also be described declaratively, SiAM-dp provides a rule-based, generic component that accomplishes this task. Thus, application developers can easily specify the transformations by a declarative approach without writing new processing components.

5.6.1 Mapping rules

In SiAM-dp it is possible to define mappings for input messages that map syntactic to semantic representations and for output messages that map semantic to syntactic representations. The definition of mappings is based on rules that consist of two parts.

Condition - The first part specifies a rule's condition and indicates when a rule should be applied. The condition is defined with the pattern model that was introduced in Section 4.4. Thus, the pattern matching mechanism uses unification on the inheritance hierarchy of the involved models. Furthermore, the pattern model allows one to assign specific content of a matching instance to variables, which may be used for the generation of the mapping target.

Target - The second part of a rule describes the content that is generated if the rule matches. This content is declaratively specified and supports the advantages of the bindable concept introduced in Section 4.3.4. Thus, it is possible to access and process content of the incoming message that has previously been assigned to variables by the pattern model. This also includes the invocation of script code or Java plugins in order to generate more sophisticated mapping results.

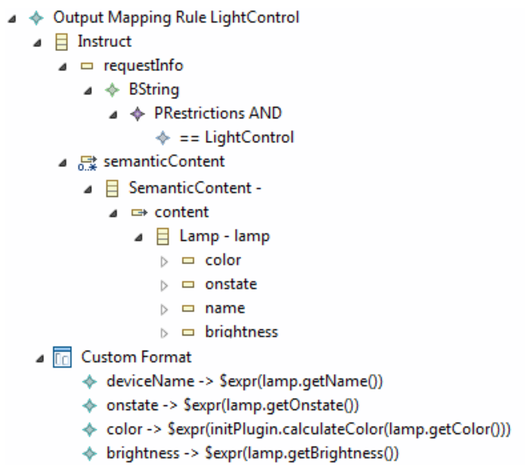
Besides mapping rules for input and output messages, it is possible to define subrules that are applied on the inner elements of a message. This helps to avoid redundancy, for example, if an inner element occurs in several distinct messages but is always mapped to the same result. The further evaluation of an inner element by subrules can be triggered in the target definition by specifying the *binding* attribute of a bindable concept to the result of the script command *Subrules.map(<content>)*. In this case the element *content* is the input for a new mapping process and validated against the conditions of all available subrules.

A rule conflict occurs if more than one rule matches an input. The set of the mapping rules in SiAM-dp is an ordered rule set. This fact is used for conflict resolution which follows the strategy *First in First Served*. This means the applied rule will be the first rule in the ordered set that is matched. Thus, the application developer has to specify during design time which rule will be applied in the case of a conflict.

5.6.2 Example

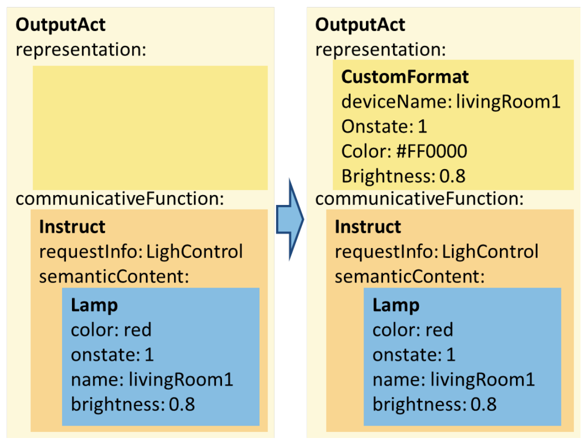
Figure 5.10 shows the example of an output mapping rule and how the application of the rule affects an outgoing message that matches the condition pattern of the rule. The pattern of the output mapping rule with the name *LightControl* describes an instruction to access the light control of the environment. An instance of the concept *Lamp* should be attached as the semantic content of the instruction.

The result of the mapping process is defined as a *CustomFormat* instance which is filled with content that is bound to the result of several script expressions. The scripts all refer to the variable *lamp* which is assigned by the lamp pattern with the *Lamp* instance of the input message. For the attributes *onstate*, *colour*, and *brightness*, the scripts simply access the get-methods of the several attributes. For the attribute *colour* the script additionally calls on a method that converts the name of the colour to its appropriate RGB value. Unlike the name of the colour, the RGB value can be processed by the target device.



```
Output Mapping Rule LightControl
  Instruct
    requestInfo
      BString
        PRestrictions AND
          == LightControl
    semanticContent
      SemanticContent -
        content
          Lamp - lamp
            color
            onstate
            name
            brightness
  Custom Format
    deviceName -> $expr(lamp.getName())
    onstate -> $expr(lamp.getOnstate())
    color -> $expr(initPlugin.calculateColor(lamp.getColor()))
    brightness -> $expr(lamp.getBrightness())
```

(a) Output mapping rule



(b) Outgoing message before and after passing the mapping rule engine

Figure 5.10 – Example of an output mapping rule

5.7 Summary

This chapter dealt with the following two research questions:

2. ***Massive Multimodality:*** *How can the massive modality of devices in a CPE be represented in a hierarchical device model and how can this hierarchy be transferred to a structured model for the representation of input and output acts in the communication between the dialogue system and devices?*
3. ***Representation of Communicative Meaning:*** *How can interaction between dialogue systems and a highly heterogeneous set of devices in a CPE be represented independently of modality, and how can this support the multimodal integration?*

In the first section the upper level of a type hierarchy for the classification of devices was introduced. The root classes are controller, sensor, renderer, and actuator. From these starting points more and more specific concepts for the description of the type and modality evolve. Orthogonal to this type hierarchy are three attributes for classification, namely the device position, interaction range, and the distinction between single and multi-user interaction, are presented. The type hierarchy is the basis of the model for the representation of dialogue acts used in Situation Adaptive Multimodal Dialogue Platform (SiAM-dp). Outgoing from an abstract class for dialogue act representation, which additionally contains meta-information like timestamps, addressee, and initiator of the message, the device specific information is structured equivalently to the abovementioned type hierarchy. Thus, devices with an equal type of content are consolidated in one class which serves as a common interface which is important for the device integration, making it easier to interchange devices without adapting the core application.

Furthermore, the model contains concepts for the modality independent representation of communicative intentions which is inherited from the standard for the semantic annotation of dialogue acts (ISO/DIS 24617). Those intentions can also be carriers of semantic content entities deployed within an interaction. Furthermore, it includes a specific model for the definition of referring expressions that are used to describe unresolved entities with relation to the current context.

The semantic representation can be loosely coupled with the device specific presentation. This approach to shift the representation of interaction to a semantic level improves the integration, expandability, and maintainability of applications since it can be used in order to specify the dialogue flow independent from the actually used modality.

Finally, a rule based mapping approach was presented that supports the mapping from syntactic to semantic representation and vice versa.

6 Declarative Specification of Multimodal Dialogue Applications

This chapter gives a detailed introduction to the models that have been developed for the declarative specification of multimodal dialogue applications. The objectives pursued with the models' concepts form the basis of the model-based development approach in SiAM-dp.

The behaviour of a dialogue application is defined with the dialogue model presented in Section 6.1. This is used by the dialogue manager to control the interaction flow of the dialogue. In Section 6.2 the model for the definition of speech grammar rules is presented. A model for describing graphical user interfaces and events on these interfaces is shown in Section 6.3. Finally, the model for project specifications is presented in Section 6.4.

6.1 Dialogue Specification Model

The dialogue manager is a core component of the dialogue system and controls the structure and interaction flow of the dialogue. It is responsible for determining the implications of user interactions and triggering system reactions. In Section 2.3.1, we discussed four kinds of dialogue management architectures that can be differentiated in complexity and powerfulness. Since the focus of this work is not on dialogue management, we followed the simplest architecture, a finite-state-manager which is a very robust and transparent solution for implementing dialogue applications. Nevertheless, other approaches like a frame-based one can be implemented with finite-state-machines, covering a broad range of multimodal interaction use cases.

6.1.1 Modelling Interaction Workflows

One of the design goals of SiAM-dp is a development approach for the rapid creation of multimodal applications for technology demonstrations or the evaluation of interaction concepts. Therefore, we want to enable even non-expert programmers to create simple applications with compelling dialogue and interaction behaviour. In the following, three concepts for the declarative specification of interaction flows are presented. In our dialogue model we adopted concepts from the latter two.

Virtual Scene Maker

An example toolkit for this kind of interaction workflow design is the *Virtual SceneMaker* (Gebhard et al., 2003, 2012) which is a visual authoring tool for virtual characters. The system's behaviour and the logical and temporal order in which scenes are played, commands are executed, and user interactions are processed, is specified by a sceneflow, a hierarchical statechart variant. The commands are expressed in a simple scripting language that supports variable assignments and function calls to the underlying higher programming language. The statechart concept in Virtual Scene Maker supports supernodes as composite states for constituting subautomaton and scoping variable validity.

Windows Workflow Foundation

Another framework is the Windows Workflow Foundation (WF) in the .NET programming environment from Microsoft (Andrew et al., 2005) which allows declarative programming with a graphical workflow editor (see Figure 6.1). With the framework's concept developers can model a program with flowcharts and simple statecharts that can call C# code fragments for more complex tasks. Assignment and condition expressions can be given by C# or Visual Basic instructions and the definition of variable scopes is supported. In other projects for customers from industry we already gained positive experience with the WF design approach that resulted in a European patent application (Bierwas et al., 2014).

State Chart XML

State Chart XML (SCXML)[1] is an XML-based markup language that describes a generic state-machine-based execution environment (Barnett et al., 2014). It is a standard published by the Word Wide Web Consortium (W3C) and is still a Working Draft specification but at a very advanced stage and near completion. The language allows one to define complex state-machines with an event-based state machine language, including sophisticated concepts like sub-states, parallel states, synchronisation, history states,

[1]http://www.w3.org/TR/scxml/

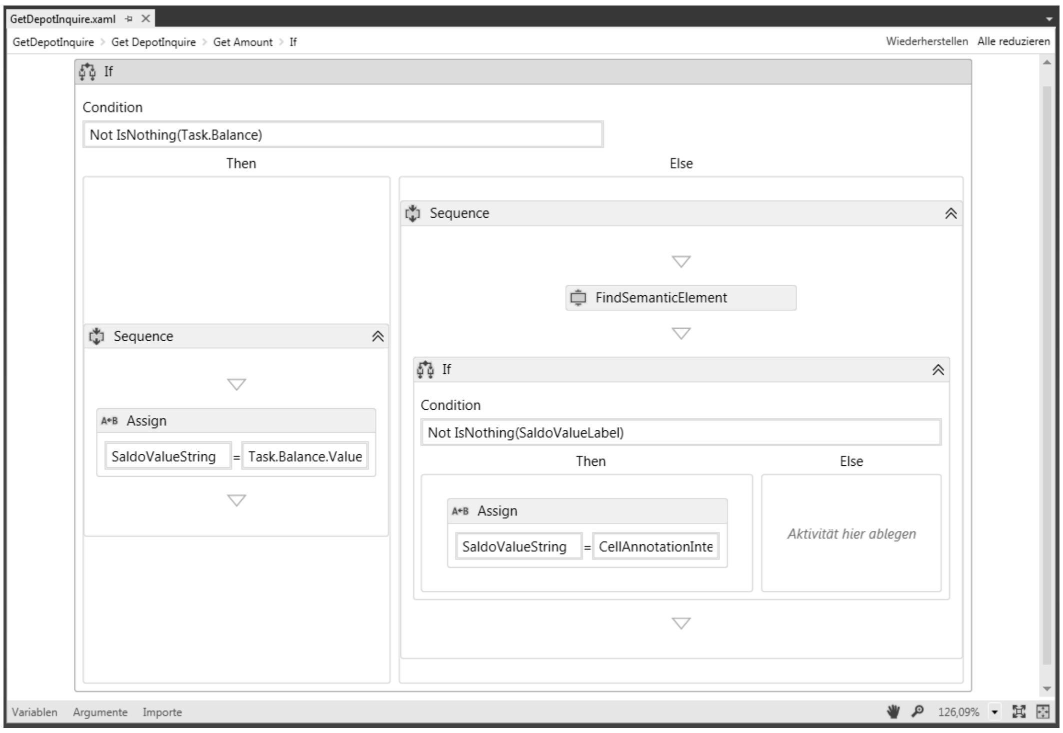


Figure 6.1 – Snippet of the graphical Visual Studios Windows Workflow editor

and concurrency. The standard was mainly developed for generalising and replacing state diagram notations already included in other XML specifications, for example, the Call Control eXtensible Markup Language (CCXML) that is used to describe telephony call control or in VoiceXML (Hoepfinger and Candell, 2010) as a high-level dialogue language. It is also planned to use SCXML as a multimodal control language in the W3C Multimodal interaction framework (MMI) (Dahl, 2013) where VoiceXML is combined with other modalities like a Graphical User Interface (GUI) described in XHTML.

The idea of statecharts goes back to Harel (1987) who extended conventional state-transition diagrams with three new features: State hierarchy, concurrency and communication. Generally, a statechart is a visual modelling approach for defining event-driven behaviour of systems that is often used in agent-based models or models for process and system dynamics. The main components of a statechart are the *states* that describe the actual context of the system and possible reactions to external events. For this, a particular state defines exiting *transitions* that are fired in reaction to a specific trigger which can be messages, events, conditions, or timeouts. After a transition is fired, the state may change and activate a new set of transitions and thus reactions. Entering states, exiting states, and following transitions can be associated with actions, e.g., output messages can be sent, the context can be changed, or methods can be invoked. The advantage of a state hierarchy is that states can be combined in composite states and share common context and transitions. Furthermore, the number of nodes and transitions and the complexity of the chart is reduced and thus the readability increased.

SCXML is supported by several implementations in diverse programming languages, like Java, C++, Python, or Java Script. One of them is *Apache Commons SCXML*, which provides a Java library for parsing SCXML documents and running them in its own engine. A main advantage of this library is the abstraction of the environment making the engine highly adaptable. In the implementation of the dialogue component in SiAM-dp the *Apache Commons SCXML* is used as the base engine for dialogue processing.

6.1.2 The SiAM dialogue model

Shukla and Schmidt (2006) start their book with the following sentences:

> "Windows Workflow Foundation (WF) is a general-purpose programming framework for creating reactive programs that act in response to stimulus from external entities. The basic characteristics of reactive programs is that they pause during their execution, for unknown amounts of time, awaiting input. . . . The focal point of the WF programming model is the concept of an activity-a program statement in a WF program. An activity's execution is inherently resumable and unfolds in an episodic manner, pausing and resuming according to the activity's interactions with external entities."

It is quite evident that this concept perfectly fits to the design of dialogue interactions. Normally, a multimodal dialogue system in a Cyber-physical Environment (CPE) is a reactive program, either reacting to the external stimuli of a user input or an event in the environment. Consequently, during an interaction, pauses with an unknown amount of time occur, for example, in an instrumented environment that automatically reacts to specific sensor events. Here, most of the time the system runs in idle and awaits events. There can also be periods of time while a user is not interacting with the system or has to decide about an appropriate answer. So current activities must be interruptible and resumable during an interaction between user and system.

Because of these arguments and the idea of creating an easy-to-use dialogue modelling approach, we decided to adopt concepts from the WF into our modelling language. This mainly concerns the main sequence control elements. Figure 6.2 depicts a diagram of the main concepts that represent these control elements in the model for dialogue definition in SiAM-dp:

`Dialogue` - This concept represents the description of the whole dialogue. It contains one *Node* which is the parent node for the complete workflow description.

`Node` - Abstract class for all control elements of the dialogue model. It is derived from the `AbstractState` concept which specifies a variable scope for the node.

`Variable` - A variable is described by the type of an entity and the name to which the entity is associated. When initialised, the variable can receive a default value.

`ExecutabeContentNode` - Contains executable content which lets the engine perform primitive actions. It allows one to modify variables, send events to other modules, raise messages, call scripts, or write logging messages (see 6.1.3).

`Sequence` - A sequence contains other nodes that are sequentially executed.

`Decision` - Enters the node in the *then*-slot if the given JEXL-condition (see Section 4.3.4) is true, otherwise the node in the *else*-slot is entered.

`While` - Repeatedly enters the node in the *body*-slot while the given JEXL-condition is true.

`DoWhile` - Enters the node in the *body*-slot once and then continuously while the given JEXL-condition is true.

`WaitForEvent` - Pauses the workflow execution until the defined event is fired and the given JEXL-condition is true.

Furthermore, it is possible to define more complex workflows with the *StateChart* concept. This node allows one to define a complete statechart that is executed when this node is reached. A statechart provides all the advantages introduced in the previous section 6.1.1, thus allowing one to define concurrent processes and global reactions to certain user events. The statechart concepts are adapted from the SCXML model and thus its complete potential is provided. The statechart concepts of the dialogue model are depicted in Figure 6.3.

`Abstract State` - Abstract parent concept for all states that primarily defines the variable scope of the state.

`State` - The base concept for the representation of a state. With the *onEntry* and

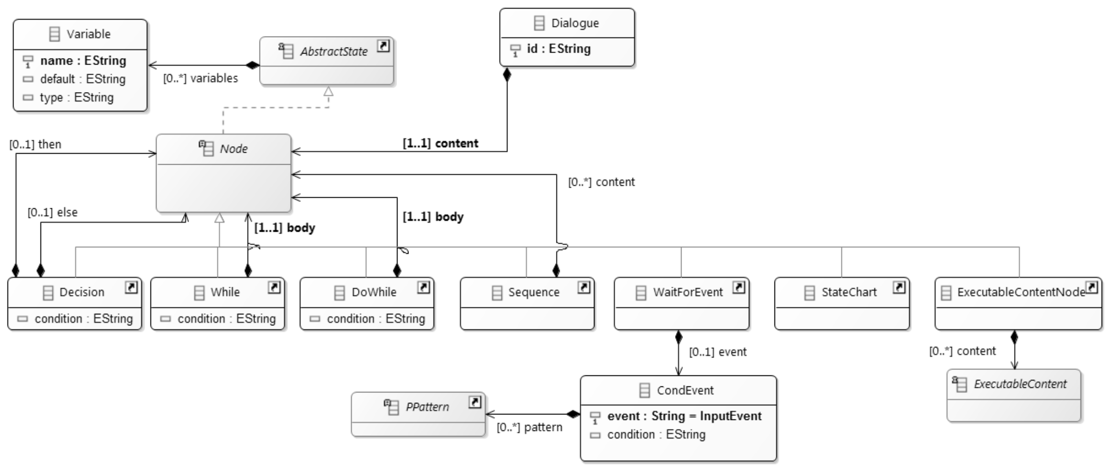


Figure 6.2 – Overview of the main control concepts of the dialogue model

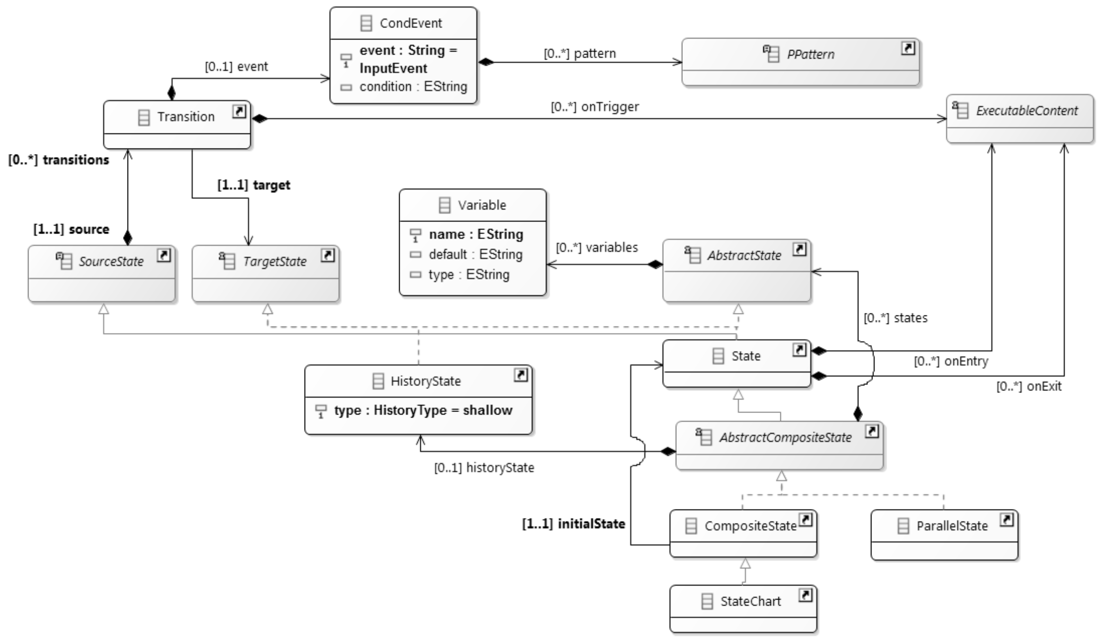


Figure 6.3 – Overview of the statechart concepts in the dialogue model

onExit slots it allows one to specify the actions that are executed when the state is entered or left, respectively. A state is derived from the concepts *SourceState* and *TargetState* and thus can be source or target content of a *Transition*.

`Transition` - Describes the transitions between states. A *CondEvents* can be assigned to a transition that specifies the trigger of the transition. The *onTrigger* slot may contain executable content, which is executed when the transition is taken.

`AbstractCompositeState` - The abstract concept represents all states that are composite states and hence can contain one or more states as content in the *states* slot. An *AbstractCompositeState* may also define a history state for state configuration recording.

`HistoryState` - A pseudo-state that records the last visited state inside a composite state before it is left. It allows one to return to a previously interrupted workflow. The *type* of a history state can be *shallow* or *deep*. A shallow history state remembers the last visited state on the same level of hierarchy. The deep history state remembers the last visited atomic descendant, which may be at a deeper level. A history state can be the target of a transition.

`CompositeState` - A composite state is a state that contains one or more sub states. The initial state that becomes active if the composite state is entered is defined with the slot *initialState*.

`StateChart` - A special version of the *CompositeState* concept, which describes the root element of a statechart. It is also derived from the concept *Node* (see Figure 6.2).

`ParallelState` - A parallel state executes all its child states in parallel. Thus, when a parallel state is active, all of its child states are active. This also means that each event is handled in each child state separately. Thus, one event can trigger distinct transitions inside the parallel child states.

`CondEvent` - This concept describes the trigger for a transition. It contains three slots. The slot *event* specifies the name of the event and corresponds to the event name in the SCXML standard. If the trigger should react to input events from the SiAM-dp event manager, the event name must be *InputEvent*. In this case it is possible to set an additional event filter by defining a pattern in the *pattern* slot. Additionally, a trigger condition can be given with a JEXL-expression in the *condition* slot. The CondEvent concept is also used for defining an event in the *WaitForEvent* concept.

`ExecutableContent` - Is an abstract concept that represents all actions from the dialogue that can be performed. An executable content can be content of an *ExecutableContentNode* or occur inside the *onEntry* and *onEntry* slots of a state or inside a transition in the slot *onTrigger*. Figure 6.4 gives an overview of the concrete concepts.

`If` - Allows the conditional execution of executable elements.

`Assign` - Changes the value of a variable in the variable scope. The *to*-slot defines the variable and the *expression*-slot contains a JEXL-expression for the value that is assigned to the variable.

`Delay` - Delays the state machine for the given amount of time.

`Raise` - Raises an internal event in the SCXML session. The identifier in the *event* slot is the event name on which a trigger can react.

`Log` - Writes a logging or debug message with the given *label* and *expression*.

`Send` - Sends an *OutputMessage* to the SiAM-dp event manager and thus enables the dialogue manager to communicate with the environment.

Since every control element is an abstract state and every statechart is a control element, it is possible to mix statecharts and control elements. If a control element is used as a state in a statechart, the instructions of the control element are initially processed when the state is entered.

In practice, the dialogue model is converted to pure SCXML by the dialogue manager component (Section 7.3) before it is provided to the Apache Commons SCXML engine.

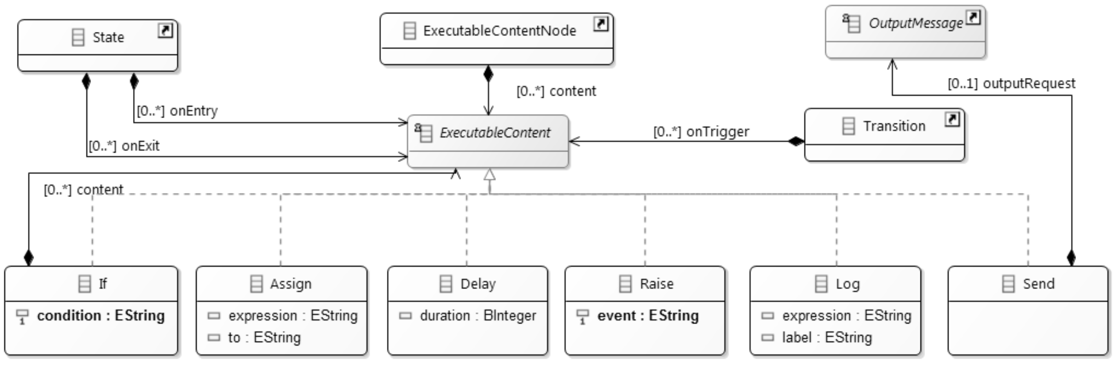


Figure 6.4 – Overview of the executable contents in the dialogue model

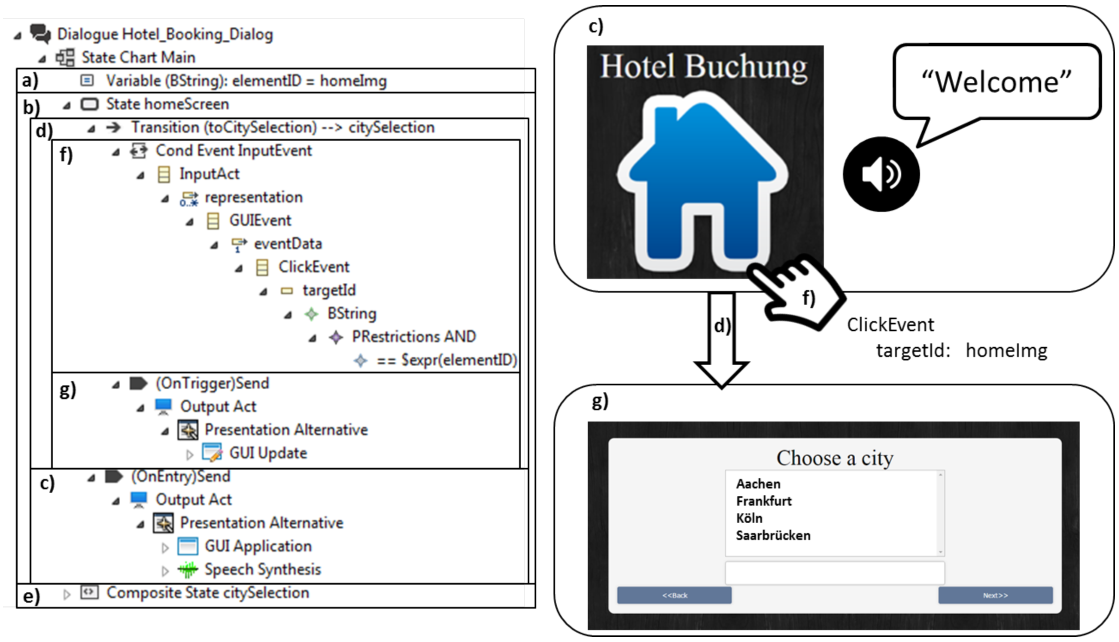
6.1.3 Embedding the IO-Model into the dialogue specification model

A focus during the development of the dialogue model lay on the integration of the input and output model that was introduced in Section 5.3. A major benefit of this approach is that application developers can directly use the model for input and output description within their dialogue specification in order to effectively and rapidly create new dialogue applications. Thus, it is possible to define patterns for input messages on which the application should react and trigger transitions between dialogue states. For this, the pattern model (see Section 4.4) is used that allows pattern matching, taking into account the type hierarchy of the involved models.

In the other direction, an application developer can descriptively define the output messages that are sent to the output devices. The abstractness of the output representation is open at the dialogue specification stage. On the one hand it is possible to just define the semantic intention of outputs using the dialogue act model (see Section 5.5). In this case the concrete output representation is planned by the presentation planning and output generation modules (see Section 7.10). On the other hand the designer can directly define the target devices and the concrete realisation of the output by using for example the GUI model (see Section6.3) or similar models for other modalities.

6.1.4 Example

Figure 6.5 shows on the left side a snippet of a simple hotel booking dialogue specification that holds a statechart. The complete statechart is the scope of the variable *elementID* (a) of type *BString* with the default value *"homeImg"*. The initial state of the statechart has the ID *"homeScreen"* (b). The actions defined in the slot *onEntry* (c) are executed if this state is entered. It contains a send command that owns an output act with two presentations in the first and unique presentation alternative. First is the definition of a GUI with a start screen, second a welcome message presented via speech synthesis. Both are sent when the dialogue is started (see the right side of the figure).



```
Dialogue Hotel_Booking_Dialog
  State Chart Main
a)    Variable (BString): elementID = homeImg
b)    State homeScreen
d)      Transition (toCitySelection) --> citySelection
f)        Cond Event InputEvent
            InputAct
              representation
                GUIEvent
                  eventData
                    ClickEvent
                      targetId
                        BString
                          PRestrictions AND
                            == $expr(elementID)
g)        (OnTrigger)Send
            Output Act
              Presentation Alternative
                GUI Update
c)      (OnEntry)Send
          Output Act
            Presentation Alternative
              GUI Application
              Speech Synthesis
e)    Composite State citySelection
```

Figure 6.5 – Snippet of a simple hotel booking dialogue. The *Send* command (c), which is executed when the initial state *homeScreen* (b) is entered, describes two output representations, a welcome page on the GUI and the speech output "Welcome". The transition pattern (f) describes a click event on the home image and is the condition for the transition (d) that updates the content of the GUI (g) when triggered.

The presented start screen shows an image which can be clicked by the user. If this happens, the screen is updated with a list of cities that can be selected. This behaviour is defined in a transition (d) with a composite state (e) as the target. The transition can be triggered by an input event (f) with a condition that restricts the triggering event to the click on the image of the start screen. This is described by a pattern for a ClickEvent with a target ID that equals the value of the variable elementID, in this example the variable's default value "homeImg" (a).

The *onTrigger* slot (g) of this transition contains the executable content *Send* that sends a new output message via the event manager. It contains an output act that updates the GUI with a new window for the city selection task. The further progress in city selection is defined in the composite state with the ID *citySelection* (e) (in the snippet not depicted in detail).

6.2 Modelling Speech Recognition Grammars

Situation Adaptive Multimodal Dialogue Platform (SiAM-dp) supports the development of speech controlled applications by providing a grammar rule model that allows one to specify speech recogniser grammars and their corresponding mapping rules to a semantic interpretation based on the semantic dialogue act model, as introduced in Section 5.5, and the domain model. A speech recognition grammar allows developers to specify the words and pattern of words that should be detected by a speech recogniser.

For the representation of the grammar we use a syntax that is closely oriented to the Speech Recognition Grammar Specification (SRGS)[2], the W3C-standard for the specification of speech recognition grammars. This standard supports the presentation of grammars in two forms, an Augmented Backus-Naur-Form (ABNF) and the XML based GRXML. While GRMXL is more often used in practice (nearly all leading commercial speech recognition systems, like systems from Nuance and Microsoft, support the grammar specification with GRXML) the ABNF form is more compact and human readable. Thus, we decided that grammar phrases are defined in the ABNF form in order to provide the application developers with a clear and compact modelling overview. The conversion from this form to GRXML is automatically done by the internal grammar management service that in a further step distributes the generated grammars to the connected speech recognisers (see Section 7.7.1).

It is possible to tag SRGS grammars with a semantic representation, the content of these tag elements is not specified by the standard. Although our system internally uses this mechanism, the grammar rule specification model has its own concepts to define mappings from recognised utterances to a semantic interpretation. These concepts are adapted to the semantic dialogue act model and are designed considering that the semantic interpretation of a dialogue act can be split into a communicative intention and the implicitly given semantic content. A linguistic utterance itself is normally given with a specific communicative intention. Thus, the complete utterance can be mapped to a communicative function.

Furthermore, the utterance can implicitly contribute semantic content to the dialogue in the form of named entities but also referring expressions. The term *named entity* is widely used in natural language processing and stands for information units like the names of persons, organisations, or locations but also numeric expressions including time, date, money, and percent expressions (Nadeau and Sekine, 2007). In named-entity recognition, the classification of the names by the type of entity they refer to is an important subtask. Since we expect that every entity of one class is mapped to one concept of the domain ontology, we support the definition of mapping rules for named entities that are similar in their semantics. In the case of a referring expression, the expression is mapped to an instance of the reference model which was introduced in Section 5.5.1.

Both the ABNF form and the XML Form of SRGS have the expressive power of a context-free grammar (CFG) and support recursive rules. However, the grammar manager of our dialogue platform allows one to enable and disable individual grammar rules and to

[2]http://www.w3.org/TR/speech-grammar/

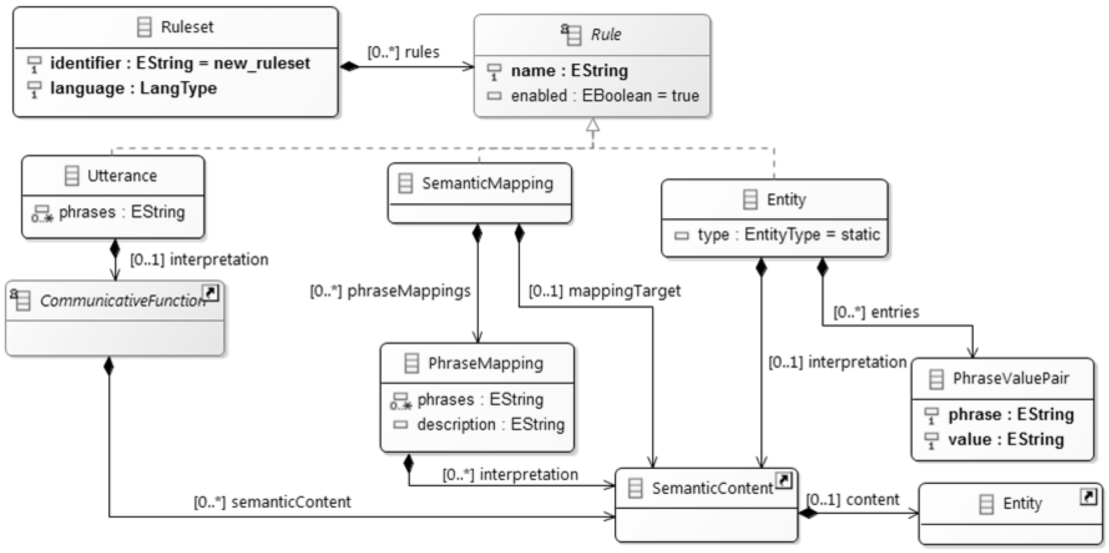


Figure 6.6 – Overview of the specification model for grammar rules

adapt named entities dynamically from the interaction flow. Thus, the currently active grammar can be manipulated based on the current discourse context.

6.2.1 Grammar Rules Specification Model

The mapping rules are modelled with the grammar rule specification model. A context free grammar is used in the sense of a semantic grammar with non-terminals (Entity Rule, Semantic Mapping Rule) that denote semantical categories. Figure 6.6 presents a diagram of the grammar model. A set of rules is collected in an instance of the concept `Ruleset`. Every rule set must have an identifier and defines the language of the grammar rules. Every rule in the rule set is identifiable by a unique name and can be enabled or disabled. The enabled state can be changed during runtime, allowing a context dependent restriction of the active rules. In the model we distinguish between three types of rules.

Utterance Rule - An `UtteranceRule` defines a set of utterances or variations of utterances that are all mapped to the same semantic interpretation. The utterance rules are located at the root level, which means that one utterance rule describes a complete speech-based input act. The phrases are defined in the `phrases` slot using the ABNF syntax. It should be noted that a rule reference that is introduced by a `$`-character refers to another rule of the rule set. Since utterance rules are only allowed at root level, the referenced rule must be either of the type `Entity` or `SemanticMapping`.

Every `UtteranceRule` can specify a communicative function as a mapping target for the interpretation. In order to integrate the interpretation result of a referenced rule as semantic content to this communicative function it is possible to specify a binding of a semantic content to this rule by setting the binding attribute to `$<rulename>`.

Entity Rule - `EntityRules` allow one to define phrases for named entities and their mapping to the corresponding semantic representation. One rule allows one to collect several named entities of the same semantic concept and to define a general rule for the semantic mapping process. Every entity rule can be of the type static or dynamic. A dynamic rule is adaptable, which means that the list of entity entries can be manipulated during runtime.

The various entries for named entities are collected in the `entries`-slot. Each entry is represented by a PhraseValuePair that consists of the phrase syntax and a value. Thus, it is possible to define different word variations for one instance, e.g., the phrases "D F K I" or "German Research Center for Artificial Intelligence" are valid expressions for our research institute. Independently from the actually used phrase, the assigned value of a named entity is definite.

The value can be used for the specification of the mapping target. This is defined in the `interpretation` slot as an instance of `SemanticContent` that serves for the semantic description of entities.

Semantic Mapping Rule - `SemanticMappingRules` allow one to define subrules that can be referred by utterance rules. With this type of rules we pursue three different objectives. The first is to avoid redundancy of phrases since subrules can be reused in several varying utterances. The second is to provide the opportunity to map single phrases directly to a semantic representation. The third is to consolidate semantic interpretations from other subrules in order to either merge several named entities into one subrule or to integrate interpretation results into more complex structures.

For every semantic mapping rule, one mapping target can be defined in the slot `mappingTarget`. This is the supertype of all semantic interpretations that are defined in this rule. The mapping target is used for the internal semantic consistency check, thus every defined interpretation in this rule must be a subtype of it. Furthermore, the rule contains a list of phrase mappings that describe the recognisable phrases and their semantic interpretation. The `description`-attribute has no practical use but serves as a documentation possibility for the application developer.

6.2.2 Example

The following example should give a better understanding of the grammar rule model. First, we introduce a named entity rule for movies as depicted in Figure 6.7. In the example two phrase-value pairs are listed, one for the movie "Iron Man", the second for

$$\begin{bmatrix}\textbf{EntityRule} & \\ \text{name:} & \text{MOVIE} \\ \text{entries:} & \begin{bmatrix}\textbf{PhraseValuePair} & \\ \text{phrase:} & \text{iron man} \\ \text{value:} & \text{1300854}\end{bmatrix} \\ & \begin{bmatrix}\textbf{PhraseValuePair} & \\ \text{phrase:} & \text{the? great gatsby} \\ \text{value:} & \text{1343092}\end{bmatrix} \\ & \ldots \\ \text{interpretation:} & \begin{bmatrix}\textbf{Movie} & \\ \text{IMDB-Id} & \$\text{expr}\left(\text{value}\right) \\ \ldots & \end{bmatrix}\end{bmatrix}$$

Figure 6.7 – Example: A named entity rule for movie names

"The Great Gatsby". The word *the* in the phrase "the great gatsby" is optional and therefore ends with a question mark. The recognisable phrases are the names of the movies. Mapping targets are the ids from the movies in the international movie data base. As the semantic interpretation for the named entities, we define an instance of the type Movie. The content of the value in the phrase mapping is mapped to the IMBD-id attribute of this instance. This is possible because the attribute is of the type BString which allows one to resolve a JEXL-expression during runtime. In this case a variable is resolved. Analogously to this rule we define a second rule for the type Book (Figure 6.8). Here we use a database ID of the book as an identifier and define a book instance as the mapping target.

Figure 6.9 shows a Semantic Mapping rule that consolidates the named entity rules for movies and books to one common upper concept, the media entity. Both semantic concepts are derived from the super type Media, which is defined as the mapping target for this rule. For each named entity rule we specify one phrase mapping. The phrases are references to the named entity rules defined before and the interpretation of each mapping is bound to the corresponding entity rule interpretation.

The advantage of defining this semantic mapping rule is that we can now specify utterance rules that refer to the `MEDIA` rule and implicitly address the more specific concepts of the entity rules. An example rule is depicted in Figure 6.10. The rule lists all syntactic variants that can be used for gaining more detailed information about a media item. Exemplarily we specify two possible phrases that contain a reference to the previously defined semantic mapping rule `MEDIA_ENTITY`. Possible utterances that match this utterance rule are:

$$\left[\begin{array}{ll} \textbf{EntityRule} & \\ \text{name:} & \text{BOOK} \\ \text{entries:} & \left[\begin{array}{ll}\textbf{PhraseValuePair} & \\ \text{phrase:} & \text{the? lord of the rings} \\ \text{value:} & 9783608938289\end{array}\right] \\ & \left[\begin{array}{ll}\textbf{PhraseValuePair} & \\ \text{phrase:} & \text{the? little prince} \\ \text{value:} & 9781853261589\end{array}\right] \\ & \dots \\ \text{interpretation:} & \left[\begin{array}{ll}\textbf{Book} & \\ \text{BookDB-ID} & \$\text{expr}\left(\text{value}\right) \\ \dots & \end{array}\right] \end{array}\right]$$

Figure 6.8 – Example: A named entity rule for book names

"What is iron man about?"
"Tell me more about the little prince!"
"What is the lord of the rings about?"

The interpretation for this rule is a communicative function, more precisely a `SetQuestion`. The aim of the question and thus the item of interest is given with *knowledgeItem* attribute. The semantic content of this question is an object of the concept `Media` which is bound to the result of the semantic mapping rule. Here it is important that the type of the content is semantically compatible to the mapping target of the referred rule.

6.3 Modelling Graphical User Interfaces

The design principle "no presentation without representation" (Wahlster, 2002) guarantees dialogue coherence in multimodal dialogue systems since the user can refer to elements of the system's output. This especially includes the representation of the current screen content on displays involved in the multimodal dialogue application. Without a dialogue system that is aware of the content it is presenting to the user, the resolution of referring expressions from speech or deictic input is impossible.

Therefore our dialogue system is always aware of the display context that is managed in a special component which is introduced in Section 7.8.1. The description of the display context is realised using a model for graphical user interfaces. In order to support arbitrary visualisation techniques, a focus lies on developing an abstract GUI model which describes the presented graphical components and contents without restrictions on the actually used devices or GUI frameworks. On the one hand, this allows one to

$$\begin{bmatrix} \textbf{SemanticMapping} & \\ \text{name:} & \text{MEDIA\_ENTITY} \\ \text{mappingTarget:} & \begin{bmatrix}\textbf{Media}\end{bmatrix} \\ \text{phraseMapping:} & \begin{bmatrix} \textbf{PhraseMapping} & \\ \text{phrase:} & \text{\$MOVIE} \\ \text{interpretation:} & \begin{bmatrix} \textbf{Movie} \\ \text{binding:MOVIE} \end{bmatrix} \end{bmatrix} \\ & \begin{bmatrix} \textbf{PhraseMapping} & \\ \text{phrase:} & \text{\$BOOK} \\ \text{interpretation:} & \begin{bmatrix} \textbf{Book} \\ \text{binding:BOOK} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 6.9 – Example: A Semantic Mapping Rule for a media entity that combines the named entity rules for book and movie

$$\begin{bmatrix} \textbf{Utterance} & \\ \text{name:} & \text{MEDIA\_INFO} \\ \text{phrase:} & \text{what is \$MEDIA\_ENTITY about} \\ & \text{tell me more about \$MEDIA\_ENTITY} \\ \text{interpretation:} & \begin{bmatrix} \textbf{SetQuestion} & \\ \text{knowledgeItem:} & \text{description} \\ \text{semanticContent:} & \begin{bmatrix} \textbf{Media} & \\ \text{binding:} & \text{MEDIA\_ENTITY} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 6.10 – Example: An utterance rule for questions about media descriptions

completely build generators that create GUI presentations only from instances of this model as it is exemplarily implemented for HTML 5 (see Section 8.3.3). On the other hand already existing GUIs may continue to be used in multimodal dialogue applications. In this case the model describes an abstract representation of this GUI with all relevant elements and content. Adaptions to the specific GUI are then only made by changing the content of the model but not the GUI component structure, which is indirectly predefined by the existing GUI.

Furthermore, the GUI model supports the creation of graphical user interfaces by providing a declarative design approach. The declarative design of user interfaces using XML structures is nowadays supported by nearly every modern GUI framework. While HTML completely relies on a declarative design, other frameworks provide a declarative specification language in parallel to the conventional programmatic approach. Microsoft developed the *Extensible Application Markup Language* (XAML) for the .NET framework in the Windows Presentation Foundation (WPF) which is an XML-based language for the specification of, among others, graphical elements, user interfaces, animations, transformations and data binding. MXML is used by Adobe Flash. JavaFX 2.0 is part of the Oracle Java distribution and, in contrast to SWT or Swing, includes the declarative XML-based language FXML for defining user interfaces. Also, the development frameworks for mobile applications are placing an emphasis on declarative GUI design. Android supports XML-based layout resources for the design of the GUI in so-called *Activity* or a component of a GUI. Apple allows one to create graphical user interfaces for iOS with the *Interface Builder*, a tool for defining graphical surfaces.

Common GUI frameworks support the philosophy of stylesheets that allow one to separate the design from the content. A style sheet is a collection of properties that can specify the look and the format of a graphical user interface, such as sizes, padding, colour, font sizes, and many more. One style sheet language in widespread use is cascading style sheets (CSS) which is applied in prominent languages like HTML, SVG, FXML and other markup languages. Hence, the concept of cascading style sheets is also supported by our model.

The next subsection gives an overview of the GUI model of SiAM-dp. First the model concepts are explained in detail. The following subsection deals with the modelling approach for GUI events. Afterwards, the way in which semantic information, communicative acts as well as semantic entities are bound to the GUI representation, is explained. The section concludes with a model example.

6.3.1 GUI Model Concepts

Figure 6.11 depicts a snippet of the GUI model diagram. The basis concept for all elements of a GUI is the concept `UIElement` which provides some relevant basic attributes like the element ID, the dedicated style, a flag that represents the enabled state, and the absolute coordinates of the element on the screen.

One important subtype of this concept is the `Container` concept which consolidates all elements that can be a container for a collection of other `UIElements`. The most elementary concrete subtype of this is a `Canvas`, an empty box that will apply an absolute layout in order to set up and place the inner elements. Elements within this container should be annotated with x and y coordinates for the position. Other concepts automatically manage the layout of their inner elements, e.g., a `VBox` arranges them vertically, an `HBox` does this correspondingly in a horizontal direction. With a `GridContainer` the developer can arrange the inner elements in a grid layout. It consists of rectangular areas defined by rows and columns, resulting in a grid that is used to position the inner items. Each element is placed in one or more cells which is fully covered by the element. `Forms` are used to combine several input elements like text fields, check boxes, radio buttons, selection lists, and more. With a `Form` it is possible to collect the information from all

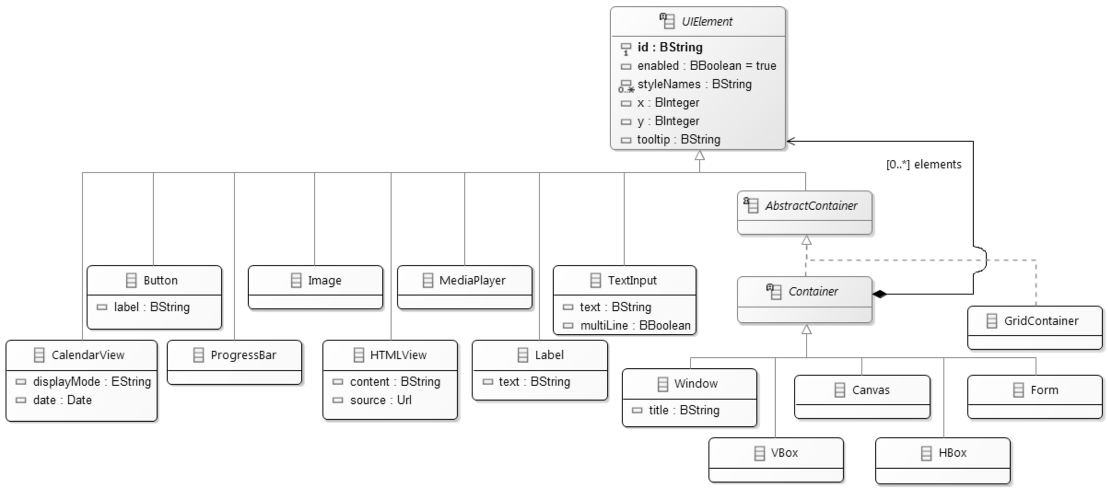


Figure 6.11 – Snippet of the GUI model

these input elements and send them together to the dialogue system in one event by clicking on a `SubmitButton`. One other important subtype is the concept `Window`, which represents the main widget of a display and is the root element of a GUI.

Another subgroup of the `UIElement` constitutes concrete GUI control elements, the figure only depicts the most common of them. However, the model supports most control elements already known from other GUI specification languages. It is possible to design labels, buttons, text input fields and areas, check boxes and more, and to configure them, e.g., by setting an attribute for the label of a button. Other elements show media information like an image or a media player. Also more complex views are possible, e.g., the model supports dialogue boxes, calendar views, progress bars, colour pickers and HTML viewer.

The content of a GUI is manipulated by the use of a `GUIRequest` (Figure 6.12), a subconcept of the IoModel concept `OutputRepresentation`. Two concrete subtypes are derived from this abstract concept. A new user interface is introduced to a display with the concept `GUIApplication`. The `GUIApplication` must define the complete GUI content with a `Window` instance. Additionally, it is possible to add a style sheet resource to the application.

If an existing GUI should be adapted, the concept `GUIUpdate` is used that contains the attribute `applicationID` for identifying the correct application to update. The following updates can be specified:

`AddUIElement -` A new element is added to the element with the given parent id. Furthermore, the position of the new element in the parent element can be specified.

`RemoveUIElement -` The element with the given ID is removed from the GUI.

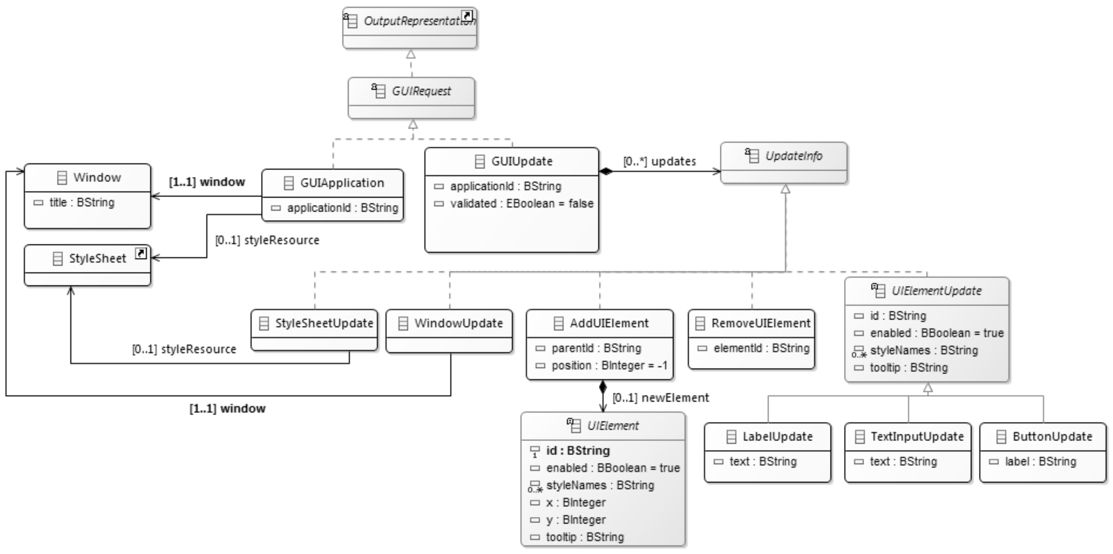


Figure 6.12 – Snippet of the model for describing GUI requests

`UIElementUpdate -` The attributes of the element with the given ID are updated to new values. Some general attributes are already defined in this abstract concept, e.g., the enabled attribute, the style, or a tooltip for the triggering of help information. Other attributes are element specific and defined in concrete update concepts for the particular types of control elements.

`WindowUpdate -` This update replaces the complete window with the definition of a new window.

`StyleSheetUpdate -` With this update the GUI can be connected to a new style sheet resource.

A `GUIUpdate` request can contain more than one update. In this case all updates are performed simultaneously.

GUI Events

A GUI is an interface with a bidirectional communication direction which means that it is not only used for output purposes, the user also interacts with the displayed control elements: He presses buttons, inserts text, or selects entries of a list and thus gives input to the application. In GUI programming, the action of a user is normally represented by GUI events in an event driven programming approach. SiAM-dp follows this approach and defines a GUI event model that describes user actions. The concept diagram of this model is depicted in Figure 6.13.

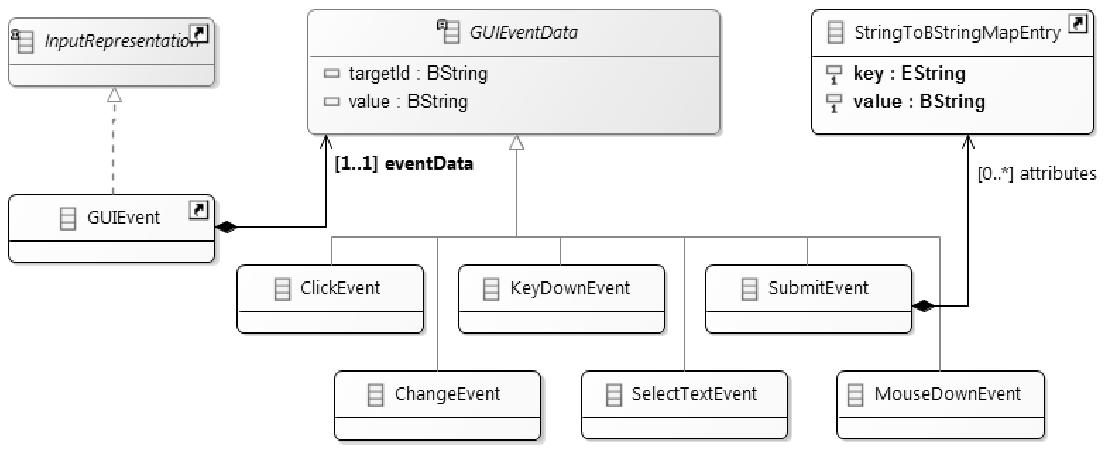


Figure 6.13 – Overview of the GUI event model

The main concept of this model is the `GUIEvent` that is derived from the IO model concept `InputRepresentation`. A `GUIEvent` contains an instance of the concept `GUIEventData` that provides all the necessary information of an event. This class is abstract and a super concept of all event specific representations. It describes some common attributes like the ID of the affected control element and a value that describes the input if necessary, e.g., an inserted text.

The particular events are represented by concrete subconcepts of the `GUIEvent` and can provide additional event attributes. The diagram shows a subset of these events. The `SubmitEvent` is a special event that is fired if the submit button of a `Form` container is triggered. It provides a map that contains all current values of the input elements inside of this form with the corresponding element ID as the key.

6.3.2 Semantic Data Binding

UI data binding is a software design pattern that simplifies the development of user interface applications by binding user interface components to entities of the application domain. One advantage is that the entity model is separated from the actually applied user interface technology and thus reusability and maintainability are ensured. Nevertheless, communication between the user interface and entity model must be possible; the user interface has to present the information of the bound data. Often the connection between the components is bidirectional which means that changes in the user interface also affect the entity model.

Actually, the GUI model supports a unidirectional connection to the entity model which means that the information presented by a control element is directly derived from the content of a bound entity. So-called adapters are responsible for the presentation based on the semantic representation of the intended output. They take the individual

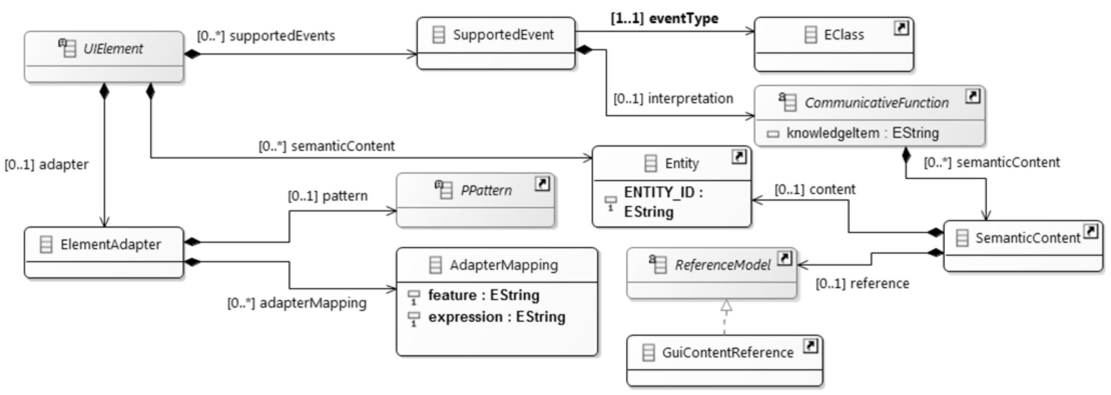


Figure 6.14 – Overview of the concepts that are involved in the semantic data binding

information elements of an entity and use them to fill the features of a control element. Figure 6.14 gives an overview of the concepts involved in the binding process:

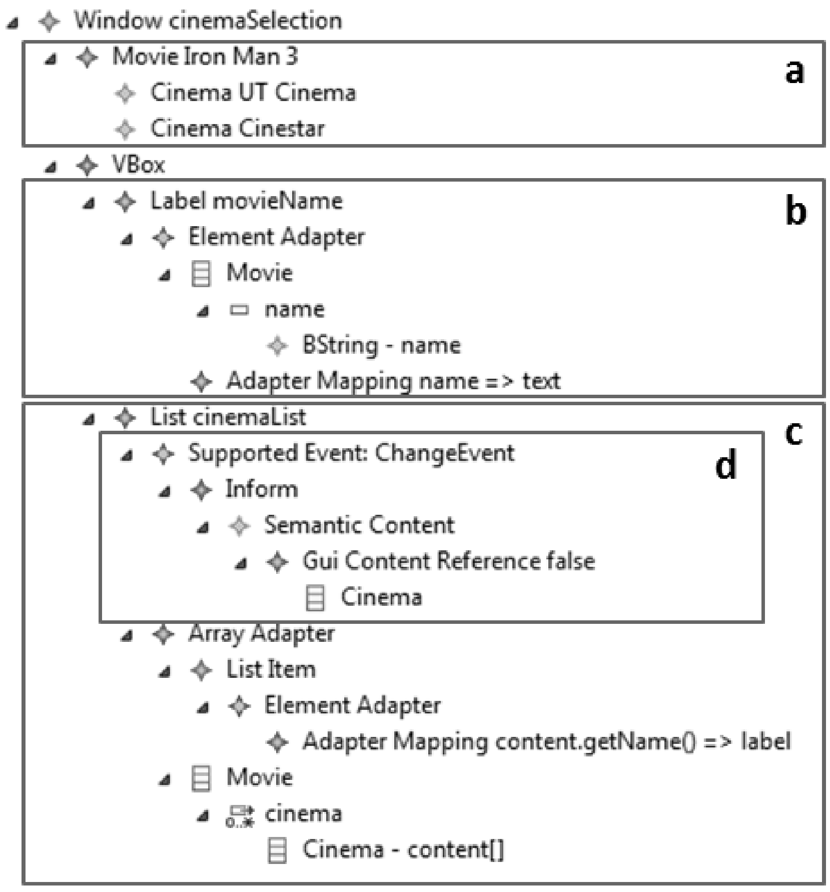
Every `UIElement` can contain the semantic content that is presented by the element which is an instance derived from the basic concept `Entity`. Additionally a `UIElement` can specify adapters that act as bridges between the `UIElement` and the contained semantic content. An `ElementAdapter` is responsible for describing the mapping between individual elements of the semantic content entities and the feature attributes of the connected `UIElement`. The individual elements are addressed with the help of a `PPattern` (see Section 4.4) and assigned to variables that are afterwards used in the specification of the adapter mappings. An `AdapterMapping` contains two attributes. The first is the target feature of the `UIElement` to which a value should be mapped. The second is a JEXL-expression that generates the new content of this feature based on the values of the variables previously assigned in the pattern. The evaluation of the mapping is processed during runtime and either triggered if a new GUI element with a binding to a semantic content is defined or if the bound content is updated.

In contrast to the other GUI languages mentioned in the introduction of this section, it is also possible to annotate control elements with the communicative intention of the user that may provoke him to interact with the element. For this, every `UIElement` can specify the supported interactions of the user with the element by the feature `supportedEvents`. Content of this feature is an instance of the concept `SupportedEvent` which describes the event type, a concept of the previously introduced event model, e.g., `ClickEvent`, `ChangeEvent`, `MouseDownEvent`. Additionally, the interpretation of this interaction can be specified by using an instance of the communicative function model (see Section 5.5). The implicitly given content of the communicative function can refer to the semantic content of the `UIElement`. This connection is declared by a `GuiContentReference` that is a subconcept of the `ReferenceModel` introduced in Section 5.5.1. The annotation with semantic information is used by the GUI input interpreter (see 7.8.2) for lifting the syntactic description of the interaction with a control element on a semantic level.

6.3.3 Example

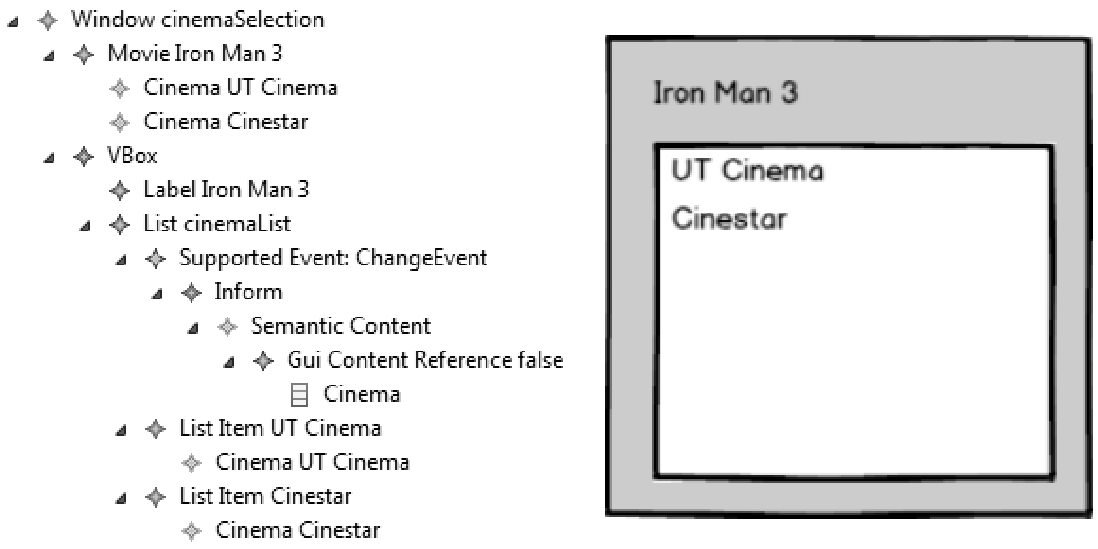
Figure 6.15 shows the example of a semantically annotated GUI model. The model instance describes a window that contains two GUI control elements and one semantic data entity. The semantic content (a) of the window is the movie instance *Iron Man 3*. The instance originally contains more data but for simplification purposes the figure only shows a list of cinemas where the movie is shown.

The control elements of the window are vertically aligned in a composite box (`VBox`). The first GUI element is a label (b) that displays the movie name. The content of the label is bound to the semantic content by an element adapter. The adapter defines a pattern describing the type of the semantic content that should be bound to the label and assigns the value of the name slot to the variable *name*. The adapter mapping describes how this content should fill the label's features. In this example the name of the movie is used to fill the text feature of the label.



```
Window cinemaSelection
  Movie Iron Man 3
    Cinema UT Cinema
    Cinema Cinestar
  VBox
    Label movieName
      Element Adapter
        Movie
          name
            BString - name
        Adapter Mapping name => text
    List cinemaList
      Supported Event: ChangeEvent
        Inform
          Semantic Content
            Gui Content Reference false
              Cinema
      Array Adapter
        List Item
          Element Adapter
            Adapter Mapping content.getName() => label
        Movie
          cinema
            Cinema - content[]
```

Figure 6.15 – A GUI model example for a window which presents a list of cinemas that show the movie *Iron Man 3*



```
Window cinemaSelection
  Movie Iron Man 3
    Cinema UT Cinema
    Cinema Cinestar
  VBox
    Label Iron Man 3
    List cinemaList
      Supported Event: ChangeEvent
        Inform
          Semantic Content
            Gui Content Reference false
              Cinema
      List Item UT Cinema
        Cinema UT Cinema
      List Item Cinestar
        Cinema Cinestar
```

Figure 6.16 – The GUI model example with resolved data bindings and the mockup of a possible rendering result

The second GUI element is a list (c) that presents the cinemas where the movie is running. The content of the list is again bound to the semantic content, in this case by an array adapter. The given pattern assigns the content of the movie's cinema slot to the variable *content*. The array adapter behaves differently from a normal adapter because it creates one GUI element for each item in an array of entities. Thus, for every cinema where the movie is running, a list item is created with the name of the cinema as a label. This is defined by the *ElementAdapter* that is attached to the *ListItem* in the *ArrayAdapter*.

With the *SupportedEvent* we define which types of user interactions are supported by the list control element. In the example (d) this is a `ChangeEvent` that fires whenever the user selects a new item in the list. The model allows one to bind the semantic interpretation of the user's intention to this event. Here the semantic entity of the cinema is introduced into the dialogue with the communicative function *Inform*. The semantic content of this communicative function is referenced to the content of the selected list item which is bound to the cinema entity it represents. This is defined by the *GuiContentReference*.

The display context manager of the dialogue platform is responsible for resolving the data bindings and complements the GUI model with the necessary information that is retrieved from the bound semantic entities. The resulting GUI model from this process is depicted in Figure 6.16. Here the control elements are extended with the text that is displayed on the screen. It must be mentioned that for every cinema a `ListItem` has been generated that contains the corresponding cinema entity as semantic content.

6.4 Project Definition Model

The previous sections introduced several models that form the building blocks of a complex multimodal dialogue application. The model that brings all these components together is the project model that is used as a starting point for the specification of a dialogue application in SiAM-dp. It comprises all information that is relevant for the declarative specification of an application, including amongst others resources, models, devices, and participating users. When the application is started the project model is read by the project manager and the contained information distributed to the appropriate components of the dialogue platform (see Section 7.4).

Figure 6.17 shows a diagram with the concepts of the project model. The root element of an instance of this model is of the concept *Project*. This concept contains references to the following elements:

`Device` - The devices that are registered and supported by the dialogue application. The platform only connects to devices in the environment that match the attributes of devices registered for the application here. Section 7.6.1 shows how the device manager controls the assignment.

`Dialogue Participant` - The participating users in the dialogue.

`Dialogue` - The model that describes the interaction flow of the dialogue manager for the application (see Section 6.1).

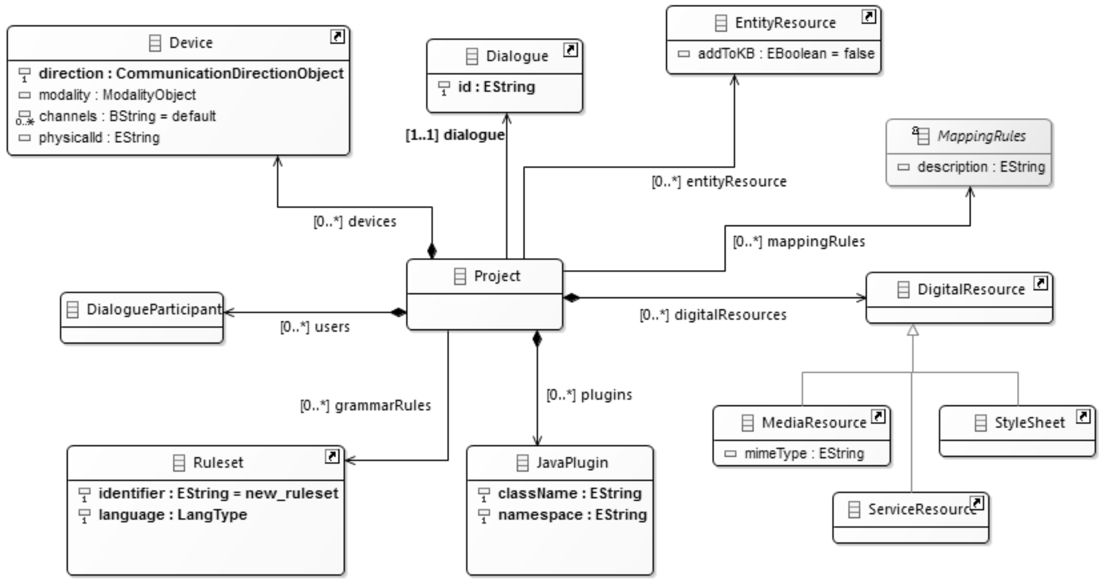


Figure 6.17 – Overview of the model for the project description of a dialogue application

`Ruleset -` The rule sets that define the grammar specifications for speech recognition (see Section 6.2). Several rule sets with distinct languages can be attached to this slot. The platform automatically selects the rule set that matches the current language configuration.

`JavaPlugin -` A `JavaPlugin` makes Java classes globally accessible from JEXL-scripts in other models via the given namespace. The class must be known from the class loader of the main application bundle and is identified by the full class name.

`Entity Resource -` The slot allows developers to specify domain entities in advance. The entities are accessible from every component in the dialogue platform by a resource management service. If the attribute *addToKB* of an entity resource is set to true, the entity directly becomes part of the knowledge base which is important for the engine that is responsible for context resolution.

`Digital Resource -` A digital resource is accessible by a Uniform Resource Identifier (URI). This can be a media resource like a video or an image, a style sheet, or the link to a web service.

`Mapping Rules -` In Section 5.6 we presented a rule-based approach for the generic mapping of syntactically to semantically represented input and output. The valid mapping rules for the project are defined in this slot.

6.5 Summary

This chapter dealt with the following research question:

4. ***Declarative Dialogue Application Design:*** *Which dialogue application specification models support the rapid development of multimodal dialogue applications in Human-Computer Interaction (HCI)?*

Several models have been designed for the declarative development of multimodal dialogue applications. These were explained in detail in this chapter. The model for dialogue specification is a combination of flowcharts and statecharts and enables application developers to directly integrate the model for input and output description. Thus, it is possible to define patterns that indicate on which input messages, transitions between dialogue states should be triggered. On the other hand developers can directly declare the output messages that are emitted. Furthermore, a very high degree of freedom for developers is ensured by the possibility to invoke script expressions and Java plugins.

The comprehensive tasks of designing graphical user interfaces and speech recognition grammars is supported by user friendly and easy-to-use models. During the development of both models, the focus was set on hiding the complexity of grammar and GUI design behind abstract concepts. Thus, they are independent from the grammar specification languages and GUI frameworks actually used for realisation. Furthermore, the models allow one to directly bind named entity rules or graphical components to the corresponding semantic entities they present. The communicative meaning behind complete speech utterances or GUI events can be represented by dialogue acts. Hereby the transformation between syntactic and semantic representation is explicitly declared in the models. The massively multimodal dialogue system can easily be extended with further modalities by utilising the generic mapping rules that were introduced in Section 5.6.

The project model consolidates all resources for dialogue applications. First of all these are instances of the previously introduced models. Other resources are the participating users, supported devices, digital resources, and semantically described entities of the world knowledge.

7 SiAM Dialogue Platform

This chapter presents the concrete implementation of SiAM-dp. The conceptual foundations have been laid in the previous chapters. The specification models for the platform were presented in Chapter 6. This chapter starts with a description of the platform architecture and an overview of the underlying technologies. In the following, the various components are described more in detail. This includes the event management (Section 7.2), session management (Section 7.6), project management (Section 7.4), dialogue management (Section 7.3), and knowledge management (Section 7.5). The technical aspects of device integration and management is explained in Section 7.6.1.

The platform additionally contains components that directly help to integrate the most common modalities. For speech recognition the creation and management of grammars and the interpretation of speech input is implicitly supported (Section 7.7). Other components, which are described in Section 7.8, are responsible for the management of Graphical User Interfaces (GUIs) and interpretation of GUI input.

Important for the massively multimodal integration is a comprehensive approach which supports cross-modal interaction in the current discourse context. On the input side this is enabled by the fusion and discourse resolution engine, which is introduced in Section 7.9. Output to the users can be situation dependently distributed to the available devices in the Cyber-physical Environment (CPE). This is explained in Section 7.10.

7.1 SiAM-dp Architecture

One of the key goals of SiAM-dp is a modular architecture that allows modules to be easily added, removed, or replaced. This is particularly important for components that integrate input and output devices since it is essential for a modern multimodal dialogue platform to simplify the extension with new devices and modalities. This is one of the main reasons for choosing the OSGi platform as underlying technology. The platform is described in the next subsection.

7.1.1 The OSGi platform

The OSGi (*Open Service Gateway Initiative*) is a specification maintained by the OSGi Alliance. It describes a Java-based platform model that supports a modular and dynamic architecture for component and service models. The multi-layered architecture is depicted in Figure 7.1

OSGi works on the Java Virtual Machine that is a platform independent interface to the underlying operating system. The following components are part of the framework:

Module Layer: Defines the basic modularization units. An OSGi *bundle* is the smallest modular unit in an OSGi service platform that can be dynamically started and stopped during runtime.

Lifecycle Management: Specifies and manages the lifecycle states of a *bundle*. A bundle can adopt the following states: *INSTALLED, RESOLVED, STARTING, ACTIVE, STOPPING*, and *UNINSTALLED*.

Service Layer: Specifies a common service model that makes services accessible throughout the system via a service registry.

Security: The OSGi specification defines an optional security layer. It is responsible for handling signed bundles and limiting execution rights for individual bundles.

Framework Services: The framework services are introduced in the core specification. They support the implementation of management components.

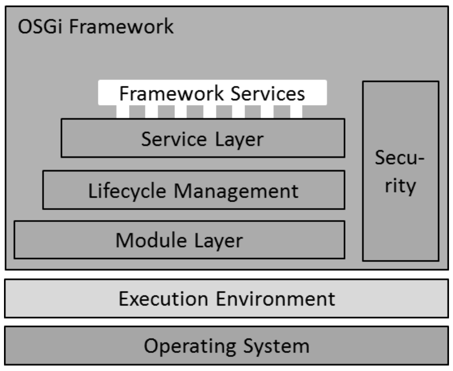


Figure 7.1 – Logical layers of the OSGi framework (according to Wütherich et al. (2009))

Besides industrial implementations of the OSGi framework also numerous open-source implementations are available, e.g., Knopplerfish, Apache Felix, or Concierge. The development of SiAM-dp is based on the Equinox OSGi framework which is promoted by the Eclipse foundation.

OSGi provides the following characteristics for the development of the multimodal dialogue platform:

Extensibility: Functionalities can easily be extended by adding new bundles.

Encapsulation: The modularization into bundles encapsulates the functionalities of the platform components and allows their extension or replacement.

Dynamic: The bundles' lifecycles are independent from each other (as long as there exist no dependencies). Thus, they can also be started and stopped independently from each other.

Dependency Management: Dependencies between bundles are declared inside the bundles' manifests. They are automatically queried and checked during runtime and development.

Orthogonal services: The service layer of OSGi gives a native support of orthogonal services.

Devices and components in SiAM-dp are defined in individual OSGi bundles. They are either part of the dialogue standard workflow or provide orthogonal services that are accessible by all other components (or both).

7.1.2 Eclipse Modelling Framework

Modelling in SiAM-dp is realised with the Eclipse Modeling Framework (EMF) which is introduced in Section 4.3.2. The basic models provided by the platform are located in the dedicated bundle `de.dfki.iui.mmds.core.model` that must be imported by the other platform components. A new domain can easily be added by creating a new EMF model bundle that extends the existing models with new domain-specific concepts. Normally, the concepts in the domain-specific model are derived from the base model which allows an easy integration of the new domain in the overall model concept of the platform. EMF provides the following advantages for the development of SiAM-dp.

- Generation of Java code from EMF models.
- Serialisation of model instances.
- Validation of model instances.
- Model editor integration into the Eclipse workbench.

7.1.3 Platform Layers

Figure 7.2 gives an overview of the main dialogue platform components. Rectangular boxes depict software components, elliptic boxes components in the physical world, e.g., sensors, devices, or actuators. The black arrows visualise the main data flow of a standard interaction. Coloured arrows represent the flow of information independent from the main data flow. All components of SiAM-dp can be attributed to one of four layers (in Figure 7.2 layers are distinguished by their colour):

Environment Layer:

The environment layer contains all modules that directly constitute the interface to the physical environment. For the interaction between human and computer it comprises **input devices** and **output devices** that exchange information with the users. Furthermore, it includes those components that build an interface to the cyber-physical environment of the current domain, e.g., a smart home or a vehicle. These components either observe changes in the environment with **sensors** or directly manipulate it by **actuators**.

It is intended that components of this layer are highly adaptable and extendable in order to develop dialogue applications for a wide range of various domains and interaction modalities. Therefore, the platform and programming language for the implementation of a module in this layer is not restricted to a specific type and the connection for communication between these modules and the core system supports a variety of different protocols, to mention some of them: TCP, REST, SIP, Apache Thrift, or the direct implementation in an OSGi bundle.

Core Layer:

The core components contain the main functionality of a dialogue application and the architecture corresponds to reference architectures well known from other multimodal dialogue system architectures, e.g., from SmartKom (Herzog and Reithinger, 2006). For all modules in the core layer SiAM-dp provides implementations with strategies that are based on the declarative models which are introduced in Section 6. These concepts are designed with respect to requirements that were derived from many years of experience in multimodal dialogue system research (Schehl et al., 2008; Porta et al., 2009a; Neßelrath and Porta, 2011; Bergweiler et al., 2010).

However, the topics for multimodal dialogue interaction are widely scattered and can have their origins in diverse areas such as research in HCI/CHI, computational linguistics, cognitive science, usability, or knowledge management. In order to provide a high flexibility and the option of customisation, all core modules are encapsulated in their own

OSGi bundles and can be arbitrarily replaced by modules that address a very specific research issue in a more adequate way.

The black arrows in the figure represent the data flow of a standard interaction. A component that is origin of an arrow provides information to the component the arrow is pointing to. Normally an interaction starts with an input from the environment layer, passes through the components of the core layer, and ends up with a response to the environment again. Nevertheless, it is feasible that when initiative is taken by the system the interaction starts in the dialogue management.

First, the input from sensors and input modalities is further processed in **unimodal interpreters** for devices and sensors, which generate a semantic representation of the input and hypotheses about the users' intentions. This step is necessary in order to make the input machine-understandable and allow further processing steps in the fol-

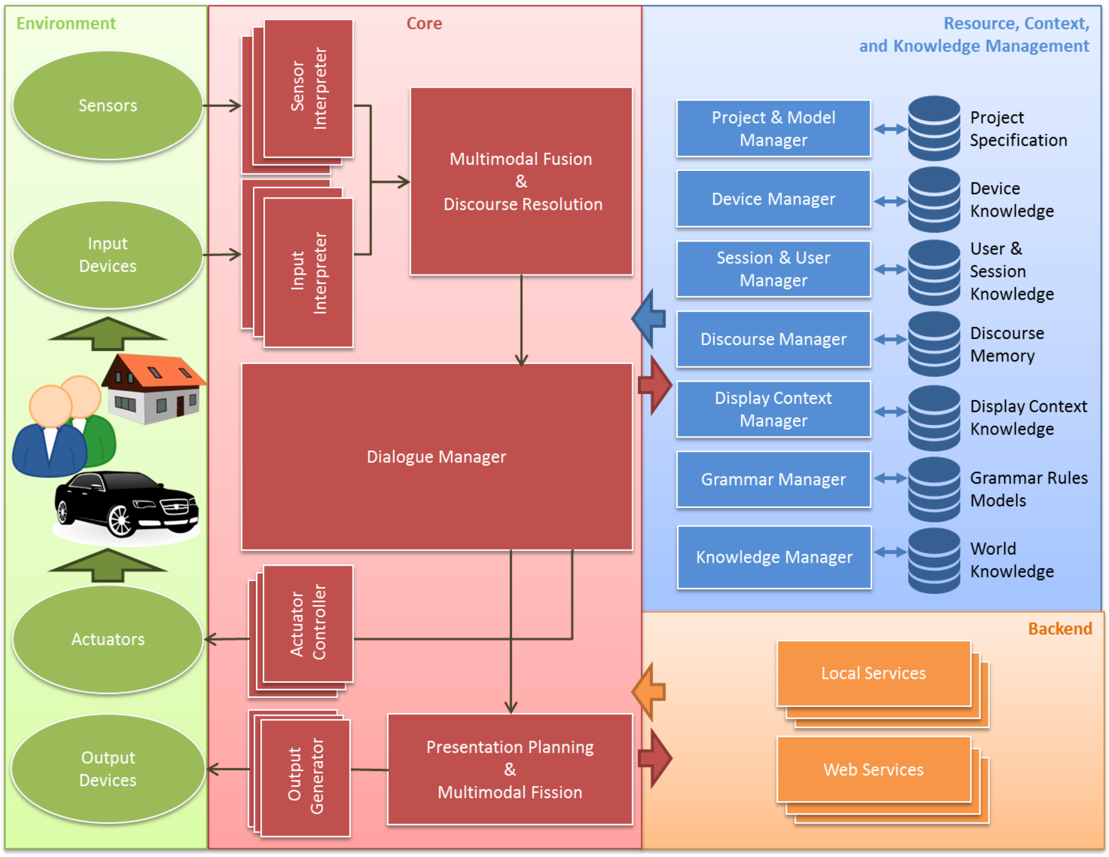


Figure 7.2 – The main architecture of the SiAM dialogue platform. The black arrows visualise the data flow of a standard interaction. The resource and context management, and backend components are orthogonally integrated and can be applied by all core components.

lowing **multimodal fusion and discourse resolution component** (see Section 7.9). It merges multimodal input from distinct modalities and complements missing and ambiguous meaning with information from the discourse context.

The main application logic is located in the **dialogue manager** (see Section 7.3) that reacts to user input and events in the physical environment. The dialogue manager is responsible for the flow of conversation with the user and generates the output of a dialogue application. If not enough information is available for performing a task, a clarification dialogue with the user may be triggered in order to retrieve additional input. Other tasks of the dialogue manager are retrieving information from backend services, which give answers to users' questions, and the control of actuators in the environment. In all cases it is an approved approach to give feedback to the user. In the dialogue manager component this is done on an abstract level using concepts for the semantic representation of communicative functions as defined in Section 5.5. So the dialogue manager decides which information is given to the user but not how and where it is presented.

The latter is realised in the **presentation planning and multimodal fission** (see Section 7.10) component that decides which devices are involved in output presentation and what the presentation looks like. Finally, the modality specific **output generators** produce concrete device and technology specific representations of the output that are presented to the user by applying the output devices in the physical environment. Additionally, **actuator controllers** generate commands in order to control actuators that change the state of the physical environment.

All modules of the core layer have access to support modules that are located in the resource and context management layer and the backend layer. These components are orthogonally integrated into the architecture by utilising the OSGi service concept. This allows to register the components to the OSGi service registry and makes them available to every other OSGi component. Thus, for every component it is possible to retrieve information necessary for dialogue processing that is provided by the support modules. Furthermore, core components can update resource and context knowledge or execute calls on external services for information retrieval and transfer.

Resource, Context, and Knowledge Management Layer:

The resource, context, and knowledge management layer comprises all components that keep track of the current system and context states. This includes a **project manager** (see Section 7.4) that manages and distributes the dialogue application specification models. The **device manager** (see Section 7.6.1) monitors connected devices and joins them to the dialogue application if the application supports them. Furthermore, it provides valuable information to the presentation planning module. The **session manager** (see Section 7.6) collects information about the active sessions, participating users, and used devices. The content represented on graphical user interfaces is managed by the

display context manager (see Section 7.8.1). The **grammar manager** (see Section 7.7) is responsible for activating and deactivating grammar rules and distributing them to the connected speech recognisers taking into account the appropriate languages. The **discourse manager** manages the discourse context (see Section 7.9.1). SiAM-dp provides an internal component that supports a semantic-based **knowledge management** (see Section 7.5).

All components are orthogonally available through the OSGi service concept. Furthermore, they can subscribe to every event that is distributed be the **event manager** (see Section 7.2) and thus have the opportunity to follow and analyse every message that is internally sent by the dialogue system.

Backend Layer:

The backend layer comprises all components that establish connections to backend systems. These can be **internet services** for information retrieval, booking, or communication but also **local running** services like a database access on the host system. All of these components are orthogonally available through the OSGi service concept.

7.2 Event Management

SiAM-dp is implemented in Java/OSGi and the native OSGi service architecture is utilised in order to enable communication between the modules and to distribute communicative messages. The black arrows in Figure 7.2 represent the communication between the dialogue platform components within a dialogue workflow. The content of the communication is based on the concept *IOEvent* as defined in Section 5.3. Subconcepts derived from this concept are *InputActs*, *OutputActs*, and control messages that change the state of a device or notify the platform about device state changes. The event messages can contain syntactic information, semantic information, or both.

Basically, every bundle in the OSGI framework can subscribe to messages that are sent within SiAM-dp. For this, the SiAM platform provides abstract event handler classes, which implement the connection to the event management service. The main component in a bundle can be derived from such handlers for input messages, output messages or any message, whichever is needed by the component. Furthermore, it is possible to use patterns as introduced in Section 4.4 in order to filter the received messages only by those that unify with the specified patterns. SiAM-dp uses a *hub and spokes architecture*, where the event management service is the hub and the single components are the spokes. A component can take the role of an *event source*, which sends events, or an *event sink*, which subscribes for specific events.

Figure 7.3 depicts how the event manager is integrated into the OSGi framework. It shows that the OSGi event admin service is exploited for message distribution within the

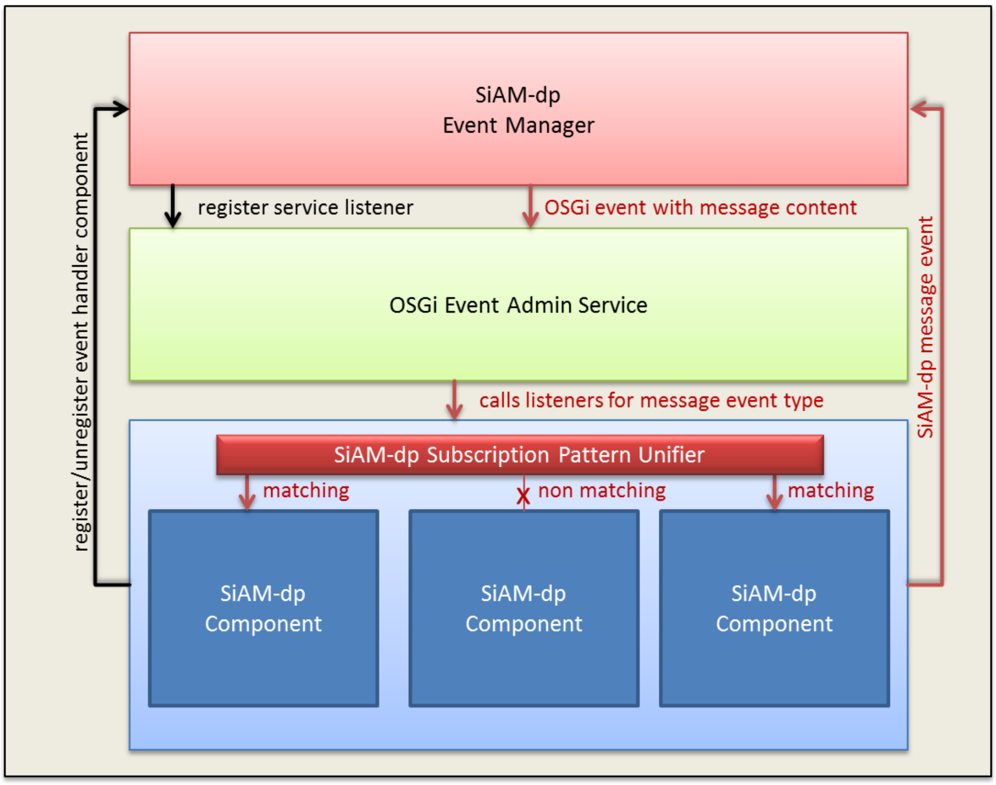

SiAM-dp
Event Manager
register service listener
OSGi event with message content
OSGi Event Admin Service
calls listeners for message event type
SiAM-dp Subscription Pattern Unifier
matching
non matching
matching
SiAM-dp Component
SiAM-dp Component
SiAM-dp Component
register/unregister event handler component
SiAM-dp message event

Figure 7.3 – Integration of the SiAM-dp event manager into the OSGi framework. The event manager distributes messages by exploiting the OSGi event admin service. The black arrows depict the handler registration workflow, the red arrows the workflow for message distribution.

platform. A reason for this is that the event admin service is a central OSGi standard service that allows to send messages across OSGi components both, synchronously and asynchronously. When sending a message synchronously the event admin waits for the response of the message receiver before the next message is sent. In contrast it directly continues with the next message if it has been sent asynchronously. The task of iterating over the registered event listeners and managing the execution queues is automatically solved by the event admin service.

Figure 7.3 contains arrows in two different colours. The black arrows show how the registration of components is realised. A SiAM-dp component can subscribe or unsubscribe to the event manager as an event listener and additionally provide a filter pattern for the messages that should be received. Then the event manager registers the handler to the event admin service and provides some additional meta information that manage the method call behaviour for preprocessor and component priority support described later in this section.

The red arrows depict the flow of a message that is sent from an arbitrary component via the event manager. The latter packs the message into an OSGi event and passes it to the OSGi event admin service which calls all registered handlers for this message. Within this call the pattern for the message content is evaluated by the subscription pattern unifier. Thus, a component's handler method is called if and only if the component's subscription pattern unifies the received message.

Normally, every component that is subscribed to a message receives a clone of the message. Thus, it it ensured that the components handle the message independently from each other and manipulations on a message made by one component have no effect on the message that is processed by another component. In contrast, in some cases it is necessary that components make adaptations to a message before other components take this modified message as input for processing. One example for this are input interpreters that enrich syntactical input messages with semantic information. This must happen before the message reaches the fusion and discourse component and the dialogue manager component. For achieving this, the event manager supports the concept of preprocessor modules. A module that is marked as preprocessor module receives a message before it is forwarded to any other component. In this case the following components receive and work with the messages modified by the preprocessor component. In order to allow more than one preprocessor component and to coordinate their temporal execution order, it is additionally possible to define priority levels of a preprocessor component. A component with a high priority is invoked prior to a component with a medium priority; likewise a component with a medium priority is invoked prior to a method with a low priority.

Figure 7.4 visualises how the distribution of messages is realised in the SiAM-dp event manager. A new message that is received by the event manager is at first sent to the preprocessor with the highest priority using the OSGi event admin service. Since it is synchronous, the call must be finished before its result is sent to a preprocessor with a medium priority. Congruently, the result of the preprocessor call with a medium priority is used for the call of preprocessor components with a low priority. The resulting message is finally cloned and the copies are distributed to any other subscribing component. In this case the calls are asynchronous.

This is for example used for the generation of output. Two preprocessor components subscribe for messages that are sent by the dialogue manager. The first one is the presentation planner, which has a high priority and determines which of the available devices and modalities are involved in the realization of the output. The second one is the dynamic mapping rule engine (see Section 5.6), which generates syntactic output presentations outgoing from a semantic representation. It requires the decisions about the involved devices from the presentation planner and thus gains a low priority. If a modality specific output generator is used, it would subscribe to the message as a preprocessor with medium priority. Thus it can generate a syntactic presentation, before the generic mapping rule engine is started.

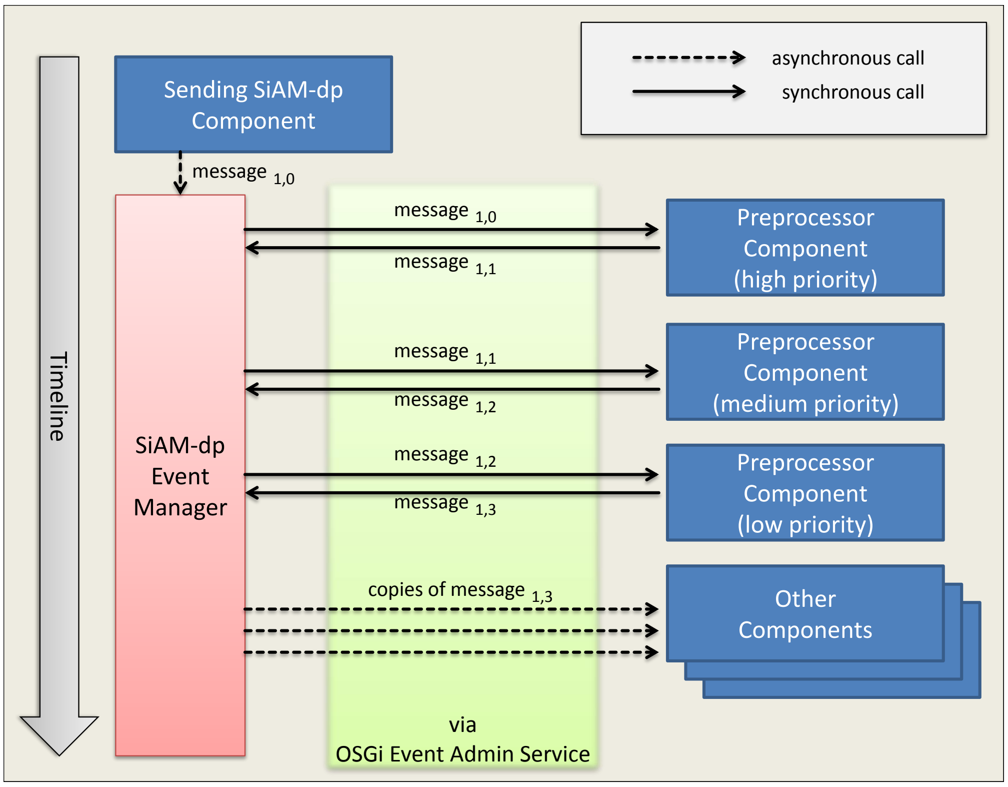


Figure 7.4 – The call sequence for components in the SiAM-dp event manager. At first a new message is sent to the preprocessor component with the highest priority. The resulting modified message is forwarded to the preprocessor component with medium priority and the result from this call again to the preprocessor component with low priority. Finally, a copy of the resulting message is asynchronously distributed to all other subscribed non-preprocessor components.

7.3 Dialogue Manager

The dialogue manager which is deployed with SiAM-dp is based on state machines and uses model instances of the dialogue model as introduced in Section 6.1 as input data. Figure 7.5 gives an overview of the internal architecture.

In the background, the Java Apache Commons SCXML[1] engine is running as an execution environment for dialogue applications. Since the expressive power of the dialogue model is not limited to pure statecharts and has been extended with further workflow specification concepts, the dialogue model is not directly compatible to SCXML. Hence, the dialogue manager contains the component *DialogueToSCXMLConverter* that receives dialogue model instances as input and generates valid SCXML documents which are the data sources for the Apache SCXML engine.

[1]http://commons.apache.org/proper/commons-scxml/

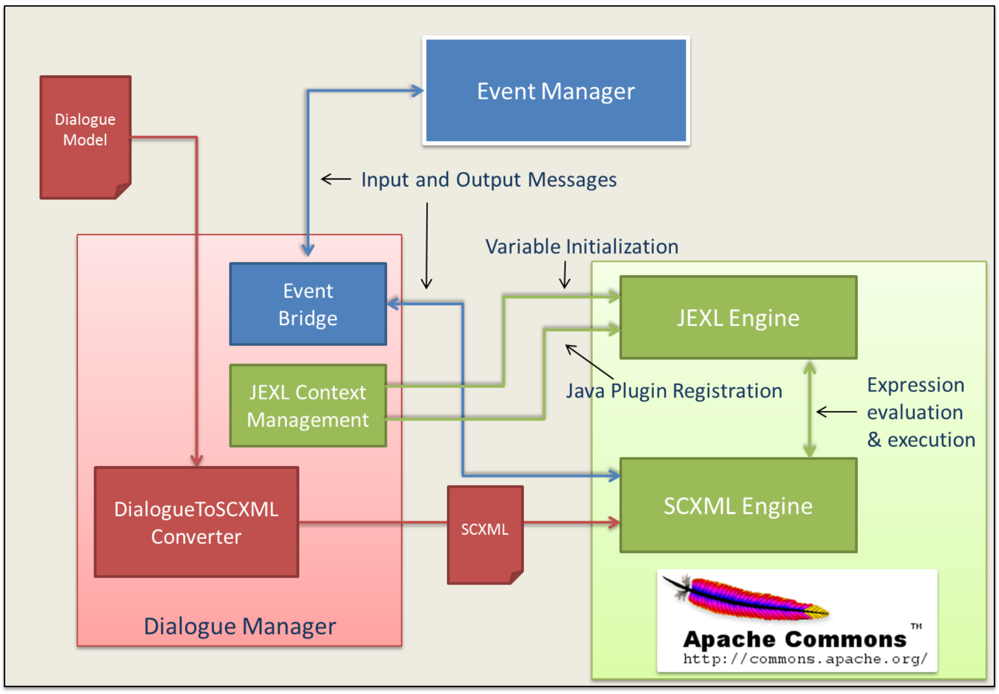


Figure 7.5 – The internal architecture of the dialogue manager. Red arrows represent the workflow of dialogue specification models that are converted to SCXML documents and forwarded to the Apache SCXML engine. Blue arrows represent the flow of input and output messages. The green arrows represent the information transfer between the Java Expression Language (JEXL) engine and other components.

Another task of the dialogue manager is to connect the underlying SCXML engine to the event management. The *Event Bridge* is responsible for this. It transforms incoming input messages from the event manager to state machine internal events. The content of the message is added as context information to these events. It is used for the pattern-based evaluation of trigger conditions in transitions of a statechart. Furthermore, parts of the message contents can contribute to the context of the SCXML engine. On the output side send-requests, which are part of the SCXML standard and are used to send events and data to external systems, are intercepted and redirected to the event manager of the dialogue system.

The SCXML specification allows to support multiple expression languages. The dialogue manager utilises the Apache Common Java Expression Language (JEXL) library[2], an open source implementation of an expression language based on some extensions to the JSTL Expression language supporting most of the constructs available in shell-script or ECMAScript. The API contributes interfaces for *Context* and *Evaluator* where the *Context* is a collection of variables that define a variable scope within a state and

[2]http://commons.apache.org/proper/commons-jexl/

the *Evaluator* is a component that is capable of parsing and evaluating expressions, the expression language engine. This API is implicitly used by the SCXML engine. Additionally, the dialogue manager can manipulate the variable context and register Java plugins to the JEXL engine that are afterwards callable from the *Evaluator*.

7.4 Project Manager

The project manager is directly launched with a dialogue application. The main task of the component is to read project specifications, to validate them, and to distribute the content to the other involved components. Furthermore, it is responsible for controlling the lifecycle of the dialogue engine and contributing services. So it is possible to start, stop, and reset the complete dialogue application with this manager.

Figure 7.6 gives an overview of the distribution flow of the resources defined in the project specification. The complete dialogue application is described with the project model (see Section 6.4). This specification is read by the project manager and the content is validated based on the EMF model specifications. In a next step the content is distributed to the components which are involved in the dialogue application. After all information is distributed, the project manager starts the dialogue engine. Additionally, it provides methods for resetting the application. The following content is distributed:

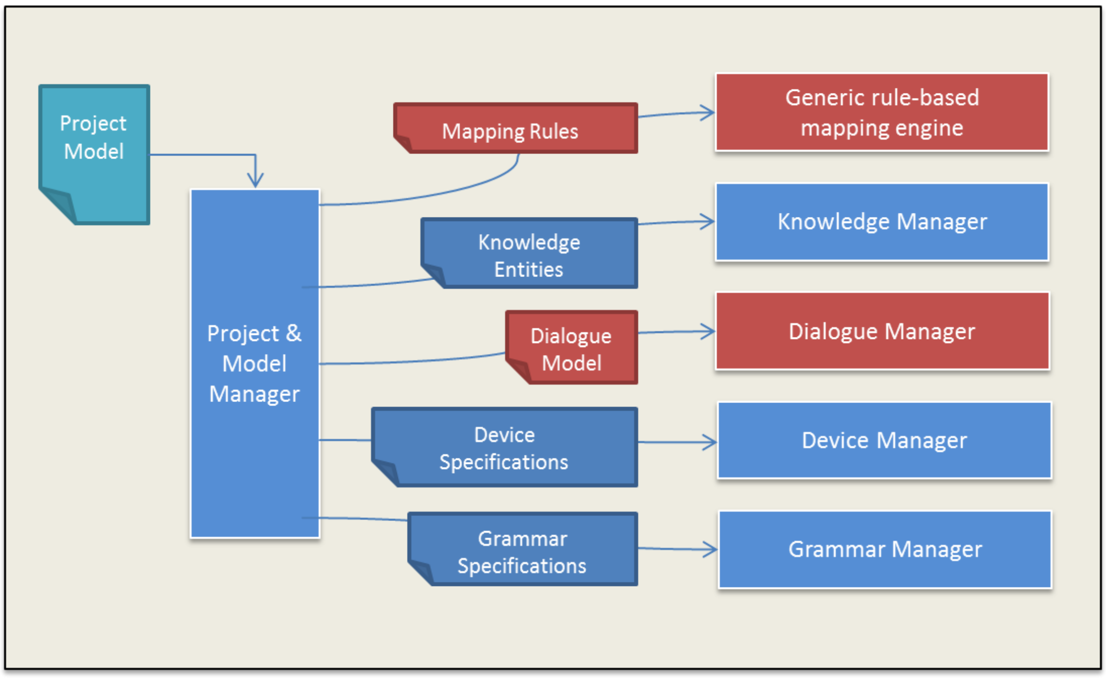


Figure 7.6 – Distribution of the dialogue specification resources by the project manager

- Predefined knowledge entities are sent to the knowledge manager and become part of the world knowledge.
- The specification of the dialogue is sent to the dialogue manager.
- The information about registered devices for the application is forwarded to the device manager.
- The specified grammar rules are forwarded to the grammar management component.
- The mapping rules for interpretation and rendering are forwarded to the generic rule-base mapping engine that has been introduced in Section 5.6

7.5 Knowledge Manager

SiAM-dp utilises an interface to a knowledge manager that encapsulates the integration of a system-specific knowledge base. Currently SiAM-dp supports an implementation for the connection to KAPCom (Knowledge Management, Adaptation, and Personalization Component), a knowledge base for scenarios supporting the knowledge exchange between the dialogue platform and other applications. The KAPCom platform has been especially used for the management of knowledge inside vehicles and the sharing of information between cars (Feld and Müller, 2011). A focus lies on the organised and efficient access to data which is required because of the heterogeneity and increased distribution of data sources. The domain-specific data is annotated with meta-information such as time, confidence, and privacy. The actual KAPCom implementation connects to devices such as the speech recogniser and speech synthesiser and retrieves information about their availability and the connection points through which they are accessible.

A newer version, which is currently in development, is started within the OSGi environment of SiAM-dp and builds on the resource management that is already provided by EMF. In SiAM-dp this knowledge manager is internally accessible by an OSGi service, which implements the abovementioned knowledge manager interface. It contains methods for the creation, modification, removal, and retrieval of knowledge entities. Furthermore, it is possible to register listeners to specific changes in the knowledge base that are notified if this change occurs. External components have access to the knowledge base via an Apache Thrift interface that is additionally provided by the knowledge manager.

7.6 Session, Device & User Management

The knowledge manager (see Section 7.5) stores information about the active sessions in an dialogue application, the participating users in these sessions, and the devices they use as interfaces to the dialogue system. Figure 7.7 depicts the model that represents these concepts and their relations to each other. A *Session* can contain several users (*DialogParticipant*), whereas a user can apply one or more user interfaces for the communication with the dialogue application. In the model, a user interface is represented by a *Service*, which, e.g., can be a gesture recognizer, a GUI, or a component for the output of synthesised speech output. One or more services are provided by a *ServiceInstance*, which describes a *Device* in the CPE from a technological point of view. This means, a service interface is one networked component in the CPE. Detailed information about the type of connection is given with the *CommunicationInfo*, respectively a communication specific subconcept of it.

In a massively multimodal dialogue system and a highly dynamic environment, the constellations represented with this model can change during runtime. External components like a device discovery service or a service for user and group detection may frequently modify this information in the knowledge base. Thus, a dialogue application must be flexible enough to adapt to newly occurring situations. The component that is responsible for monitoring this information is the session and user manager that collaborates with the device manager (see Section 7.6.1). It is also responsible for updating the session and user model with information which is retrieved from the currently running dialogue application.

7.6.1 Device Manager

For a massively multimodal dialogue system, it is important that the system is aware of the services of the existing and available devices. In SiAM-dp the device manager is responsible for this, a service that keeps track of the registered, discovered, and connected

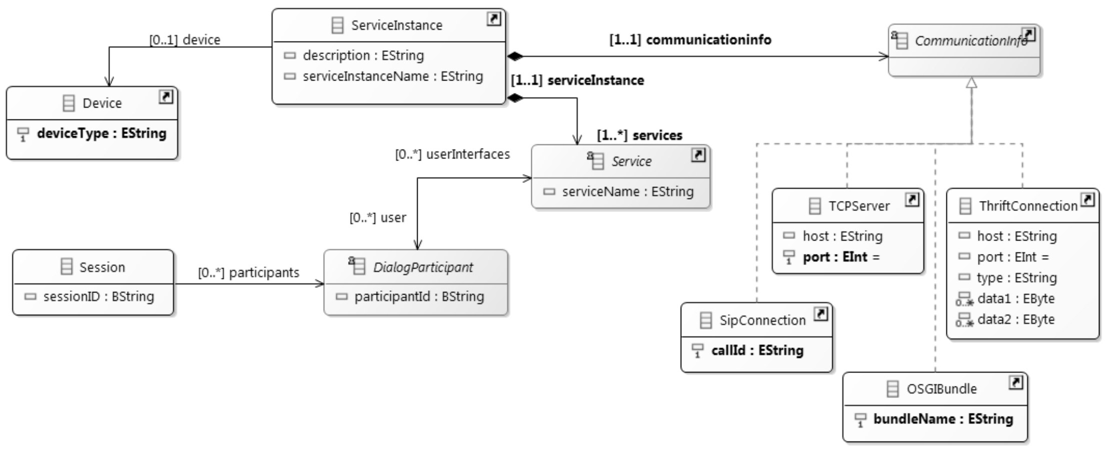


Figure 7.7 – The concepts for the representation of sessions, users, devices, user interfaces, and their relations to each other

services (user interfaces). The supported user interfaces of a multimodal dialogue application are declared in the project specification (see Section 6.4) with information about the modality, type of the interface, service instance ID, and service name. Based on this information, a matching process is started that compares the registered user interfaces with the discovered user interfaces, which are available via the knowledge manager.

Device Assignment

The device manager uses three different sets for the device management:

Registered User Interfaces: These are user interfaces that are registered in the project definition and thus are supported by the dialogue application. Additionally, new user interfaces can be registered and existing user interfaces can be unregistered during runtime.

Discovered User Interfaces: The discovered user interfaces are not directly maintained by the device manager. Information about available user interfaces are stored in the knowledge base and can externally be manipulated, e.g by a device discovery component. This means that user interfaces can be discovered and lost during runtime. The device manager uses this information and is also registered as listener to the knowledge base and thus is notified about changes.

Connected User Interfaces: User interfaces that are actually connected and involved in the dialogue application.

During runtime not every registered user interface must be available and not every discovered user interface must be connected to the dialogue application. In fact, the device manager permanently compares the set of discovered user interfaces with the set of registered user interfaces. If a user interface is discovered that meets the requirements of a registered user interface, the dialogue application connects to this user interface and makes it accessible to the dialogue application.

The matching process between registered and discovered user interfaces is divided into two steps. In the first step, it checks for an explicit assignment of the user interface, which is given by the service instance ID and the service name. If no explicit information is available, the matching process is based on the given meta-information of the user interface. This includes attributes describing the modality, the addressed user, and the type of the user interface (see Section 5.4). Thus, it is possible to register a user interface to a dialogue application independently from the concrete applied one, making a dialogue application adaptable to the user interfaces currently available in the CPE.

7.7 Speech Recognition Components

SiAM-dp contains an integrated speech recognition component that is responsible for the generation of speech recognition grammars and the semantic interpretation of speech input from connected speech recognisers. The component is working on instances of the grammar rule model that has been introduced in Section 6.2. This model provides an integrated and well structured approach for the definition of utterance phrases, sub phrases, and entities and allows to annotate them with semantic interpretations.

Figure 7.8 gives an overview of the elements in the speech recognition component. The **Grammar Management Service** (see Section 7.7.1) is responsible for managing the registered and active grammars and for distributing grammar updates to the connected speech recognition devices. For this the **GRXML Converter** (see Section 7.7.2) first generates valid GRXML syntax from the active grammar rules.

The result of a speech input from speech recognisers is first processed by the **Speech Recognition Interpreter** (see Section 7.7.3) before it is forwarded to other components via the event manager. Here, the input is raised on a semantic interpretation level based on the annotations in the grammar rule model.

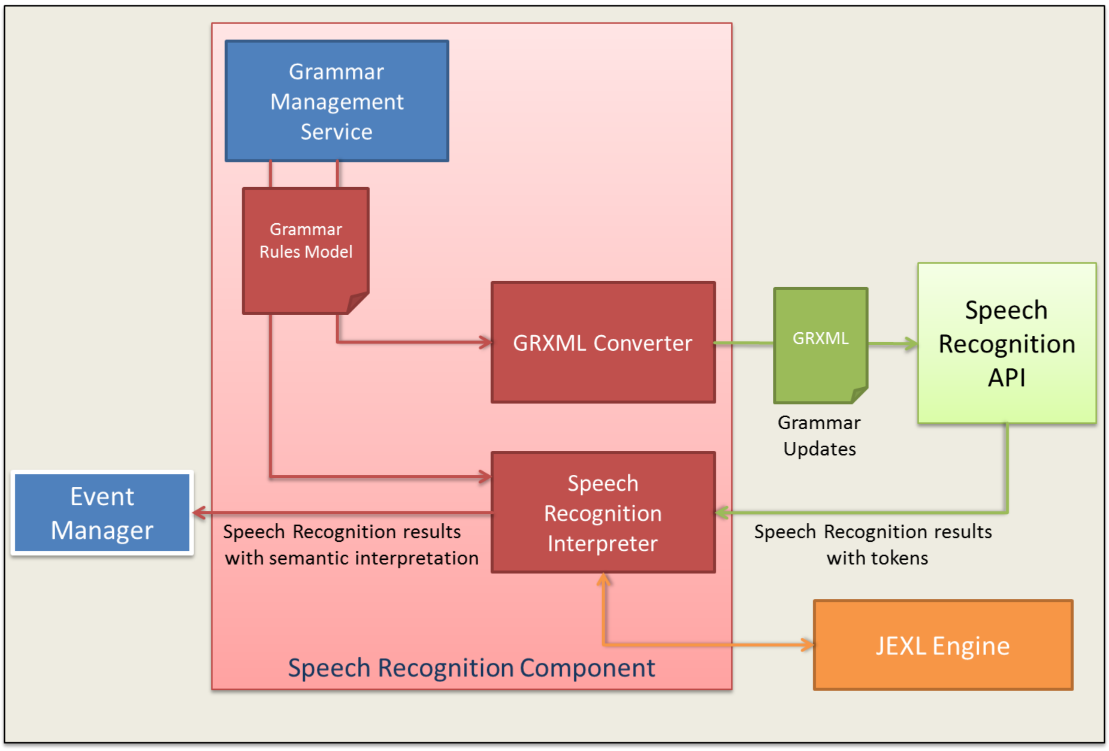

Grammar Management Service
Grammar Rules Model
GRXML Converter
GRXML
Grammar Updates
Speech Recognition API
Event Manager
Speech Recognition Interpreter
Speech Recognition results with semantic interpretation
Speech Recognition results with tokens
JEXL Engine
Speech Recognition Component

Figure 7.8 – Detailed overview of the speech recognition component

```
public interface IGrammarManagementService {
        // rule management
        void addRuleset(Ruleset ruleset);
        void disableRule(String rulesetID, String ruleName);
        void enableRule(String rulesetID, String ruleName);
        void enableRuleset(String rulesetID);
        void disableRuleset(String rulesetID);
        Rule getRule(String rulesetID, String ruleName);

        // manipulation of dynamic entities
        void setDynamicRuleEntities(String rulesetID, String ruleName,
                PhraseValueList entities);
        void addEntityToDynamicRule(String rulesetID, String ruleName,
                String phrase, String value);
        void clearDynamicRuleEntity(String rulesetID, String ruleName);

        void sendGrammarToDevice(Device device);
        void commit();

        void reset();
}
```

Listing 7.1 – The grammar manager interface

7.7.1 Grammar Management Service

The registered grammar rules can be modified during runtime. Thus, it is possible to enable or disable single rules and even complete rule sets. Furthermore, rules that specify lists of entities can be extended with new content, e.g., from a backend service. All this functionality is covered by the grammar management service. Listing 7.1 presents its interface.

The first set of methods provides access to the registered rule sets and allows one to enable and disable them. Thus, the grammar space can be reduced depending on the context, which improves the recognition accuracy. The second set can be used for editing dynamic entity rules. Furthermore, a grammar can manually be sent to a specific connected device. If some changes to the active grammar rules are committed with the corresponding method, the step of grammar distribution to the proper connected speech recognition devices is processed automatically.

A further task of the grammar manager is to deploy grammars to newly connected speech recognition devices. For this the grammar management service is informed by the device manager (see 7.6.1) if a new speech recognition device is connected and provides the active grammars to this device.

7.7.2 GRXML Converter

The most speech recognition engines use standardized GRXML as grammar definition syntax. Thus, the grammar model must be converted to valid GRXML syntax. The GRXML Converter component is responsible for this. It has the following tasks:

1. Parsing the ABNF syntax that is internally used in the grammar rule model.
2. Validating the syntactical correctness of the given ABNF.
3. Converting the ABNF syntax into GRXML.
4. Representing the references between the specified grammar rules in GRXML.
5. Creation of annotation tags in GRXML that help to reconstruct the interpretation of a speech input.

Tags are grammar rule expansions, which are arbitrary strings that may be included inline in each rule. They do not affect the grammatical patterns for speech recognition. Moreover, they provide additional content that can be used for semantic interpretation. We use this annotation mechanism for providing the rule ID of these rules that have been consulted for the recognition of a speech utterances. For the speech input interpreter this is a very valuable information and used for the reconstruction of the semantic interpretations which the grammar rules are annotated with.

7.7.3 Speech Recognition Interpretation

The supported semantic annotations in the grammar rule model cannot directly be integrated into the generated GRXML syntax since here only string content is supported for the interpretation of speech input. Thus, the result of speech recognisers, that can only use GRXML as input and have no access to the defined semantic annotations, does not contain the semantic interpretation of a speech input act we defined in the grammar rule model.

As mentioned in the previous section, we use GRXML tags in order to identify the rules and referenced subrules that contain the word patterns for the recognised utterance. This information is provided together with a recognition result. Listing 7.2 gives an example for this. The speech recognition interpreter combines this recognition result with the information from the grammar rule model and reconstructs the semantic interpretation based on the semantic annotations that are attached to the involved grammar rules.

For this, it retrieves the semantic annotation of the root rule and resolves its inner content (compare the grammar rule annotation example in 6.2.2). During this resolution process, it has to follow the references to other grammar rules and integrate their semantic annotations to the interpretation of the root rule. Furthermore, JEXL-Expression that can be used for generating attribute content must be evaluated during runtime by accessing the Apache JEXL-engine.

```
<hypotheses confidence="0.9545653" grammar="ROOT" utterance="what␣is␣iron␣
    man␣about" xsi:type="io:SpeechHypothesis">
        <tokens key="RULE_ID" value="sab_demo@MEDIA_INFO"/>
        <tokens key="text" value="what␣is␣iron␣man␣about"/>
        <tokens key="MEDIA_ENTITY.RULE_ID" value="sab_demo@MEDIA_ENTITY"/>
        <tokens key="MEDIA_ENTITY.text" value="iron␣man"/>
        <tokens key="MEDIA_ENTITY.score" value="1"/>
        <tokens key="MEDIA_ENTITY.MOVIE.RULE_ID" value="sab_demo@MOVIE"/>
        <tokens key="MEDIA_ENTITY.MOVIE.text" value="iron␣man"/>
        <tokens key="MEDIA_ENTITY.MOVIE.score" value="1"/>
        <tokens key="MEDIA_ENTITY.MOVIE.value" value="1300854"/>
        <tokens key="MEDIA_ENTITY.PHRASE_INDEX" value="0"/>
</hypotheses>
```

Listing 7.2 – Example for a hypothesis in a speech recognition result. The tokens provide additional information about the ID, recognised text as well as the recognition score for the main grammar rule (MEDIA_INFO) and all involved subrules (MEDIA_ENTITY, MEDIA_ENTITY.MOVIE).

7.8 Managing Graphical User Interfaces

SiAM-dp contains a component that manages the display context of GUI presentations. Since context knowledge about information that is presented to the user on a graphical display is highly important for the resolution of deictic expressions, spatial expressions, and cross-modal references, it is necessary to manage the internal representations of GUIs and make them available to other components that are involved in the dialogue application. The component works on GUI model instances, which are in detail introduced in Section 6.3. This model provides an integrated and well structured approach to declaratively create GUIs. Furthermore, it allows to bind semantic content to GUI elements and to annotate GUI events with a semantic representation in form of dialogue acts.

Figure 7.9 gives an overview of the elements and the information flow in the GUI management component. The **Display Context Management Service** (see 7.8.1) is responsible for managing and updating the active display contexts and distributes the modified GUI model instances to the connected GUI rendering engines. These are responsible for the creation of the concrete presentations based on the platform and technology supported by the engine. Furthermore, they have to report user events back to the dialogue system based on syntactic representation. The **GUI Input Interpreter** has the task to shift the syntactic representation on a semantic level. For this, it evaluates the semantic annotations for content and input interpretation that are attached to the GUI model.

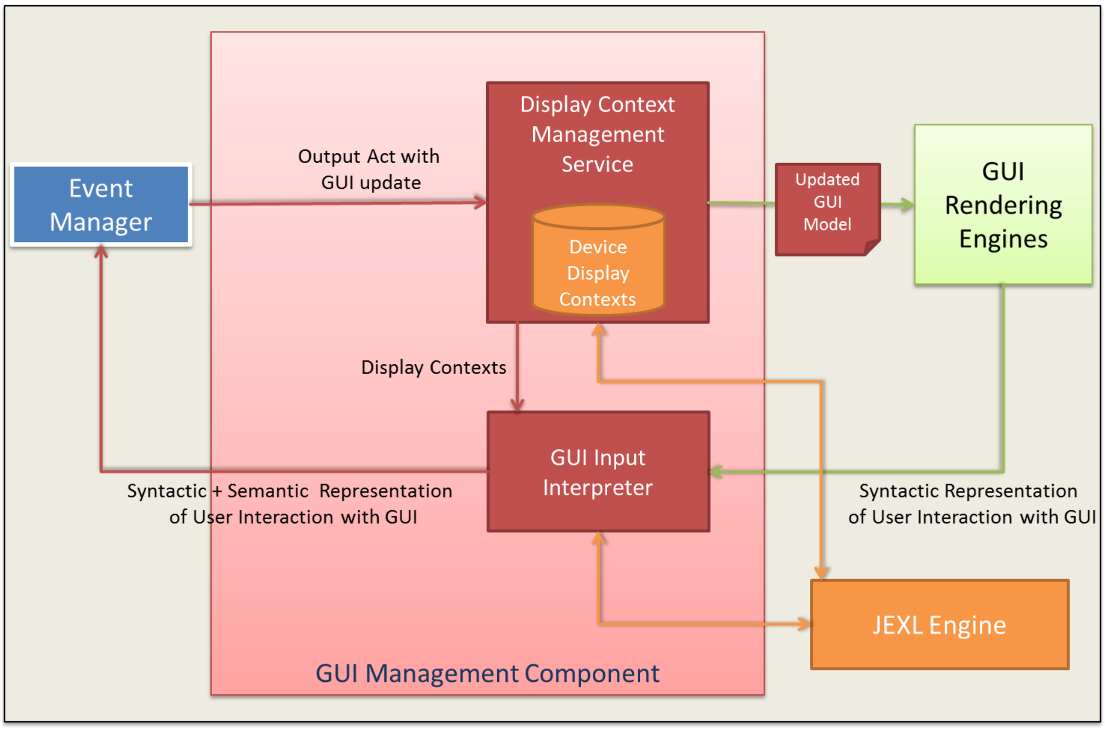

Event Manager
Output Act with GUI update
Display Context Management Service
Device Display Contexts
Updated GUI Model
GUI Rendering Engines
Display Contexts
GUI Input Interpreter
Syntactic + Semantic Representation of User Interaction with GUI
Syntactic Representation of User Interaction with GUI
JEXL Engine
GUI Management Component

Figure 7.9 – Information flow in the GUI management component

7.8.1 Display Context Manager

The display context manager contains all GUIs that are currently presented by the dialogue application. Thus, the manager holds one appropriate GUI model instance for every GUI that is presented by a graphical output device. The GUI update mechanism supports two types of output requests: it can specify a complete new GUI application and provide the complete GUI model that is presented. Or it can update the content of an already existing GUI model instance. In the first case the old display context of a device is replaced by a new one. In the second case the display context manager is responsible for adapting the actual display context with the changes committed with an update request.

Before it distributes a GUI model to a connected GUI rendering engine, the display context manager has to resolve semantic data bindings and to complement the actual GUI model with the necessary information that is retrieved from the bound semantic entities (compare section 6.3.2). Thus, a GUI model instance also changes if a new semantic entity is bound to a GUI element. Furthermore, the display context manager has to evaluate content that is computed during runtime by accessing Java Script expressions or executing Java plugin calls. For this the Apache JEXL engine is called.

7.8.2 GUI Input Interpreter

Since a concrete realisation of a GUI does not contain information about the semantic annotations in the GUI model, we need a component that complements the interpretation of the syntactic representation of a user event with a semantic representation. The GUI input interpreter fulfills this task and is registered as a preprocessor for incoming GUI events. Together with the semantic annotations in the GUI model instance, which it retrieves from the display context manager, it reconstructs the annotations about the user's intention. Listing 7.3 shows an example of the syntactic representation of a GUI input for the GUI model instance example in Figure 6.16. The GUI input interpreter preprocesses this message and gets the current display context for the device with ID *car_display* from the display context manager. In the next step, it searches for the Element with ID *"cinemaList"*, which besides a semantic annotation of the *ChangeEvent*, contains a list of items with *Cinema* entities attached as semantic content. In a last step it adds the communicative function, the *ChangeEvent* is annotated with, as a hypothesis to the InputAct. The containing *GuiContentReference* refers to an entity of type *Cinema*. This is taken from the ListItem with index *0* and added to the communicative function. The result is shown in listing 7.4.

```
<io:InputAct>
  <representation device="car_display" modality="\gls{gui}" xsi:type="gui:\
      gls{gui}Event">
    <eventData index="0" targetId="cinemaList" value="Cinestar" xsi:type="
        gui_events:ChangeEvent"/>
  </representation>
</io:InputAct>
```

Listing 7.3 – Example for a GUI input event

```
<io:InputAct>
    <representation ...
    <hypotheses confidence="1.0">
        <communicativeFunction xsi:type="communicative_functions:Inform">
            <semanticContent resolved="true" validity="-1">
                <content name="Cinestar" xsi:type="sab:Cinema"/>
                <reference xsi:type="references:GuiContentReference">
                    <referencePattern instIdx="-1">
                        <type href="http://www.dfki.de/iui/mmds/sab_demo/
                            model#//Cinema"/>
                    </referencePattern>
                </reference>
            </semanticContent>
        </communicativeFunction>
    </hypotheses>
</io:InputAct>
```

Listing 7.4 – GUI input event after passing the GUI Input Interpreter

7.9 Fusion & Discourse Resolution

The SiAM-dp platform contains a component which can be used for handling multimodal and dialogue phenomena, the Fusion and Discourse Engine (FADE). Our FADE component is inspired by the work of Pfleger (2007) and supports the core functionalities described by him. Section 5.5 already describes how his ontological modelling approaches for dialogue acts and referring expressions have been adopted to the general modelling environment of the SiAM platform.

FADE provides generic and reusable implementations for the resolution of missing content that arises from multimodal integration and phenomena in dialogue. It monitors the incoming and outgoing dialogue acts during discourse and becomes active in three situations:

1. An input act contains unresolved semantic content.
2. An input act is marked as ellipsis.
3. An output act contains a referring expression to the long-term memory.

Furthermore, it is responsible for updating the discourse context with the monitored information and works closely together with the discourse manager. Hence, in the next subsection the internal representation of data in the discourse context is described.

7.9.1 Managing the Discourse Context

The discourse context provides the required information for resolving context dependent phenomena in the FADE component. Furthermore, it can be used by other components for arbitrary analyses on the discourse since it keeps track of the complete dialogue history. Internally, three different types of data are managed in their own data structures:

Discourse History: In the discourse history the complete communication between the dialogue platform and the CPE is logged. Every input and output act is saved in a list ordered by the time point when the action occurred.

Long-Term Memory: According to Pfleger (2007) the Long-Term Memory (LTM) is used for representing the knowledge about the world independently from the knowledge that has implicitly been mentioned during the discourse. This has two reasons: First, it is possible to reference to entities in the world that have never been mentioned during the previous discourse. Second, it serves as source for the real world context since content that is introduced by discourse not necessarily describes the real world state but rather the subjective point of view of the user. The long-term memory is not implicitly modified by the FADE component. Moreover, entities must be explicitly modified by, e.g., external services that collect environment information or the dialogue management. For this, every entity in the LTM is

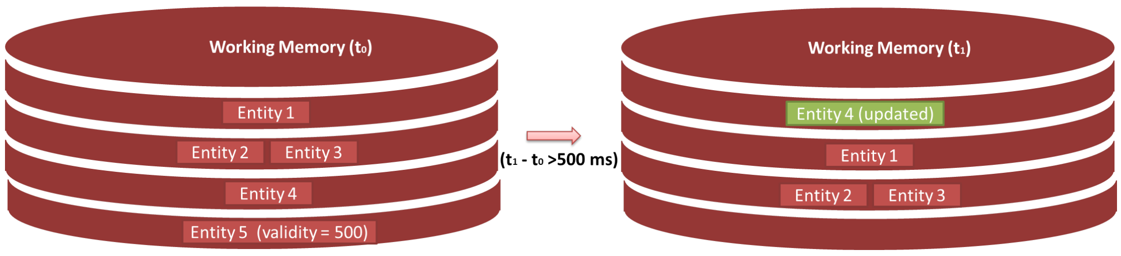


Figure 7.10 – Structure of the working memory. The resolved semantic content entries of every communicative act are saved in an own entry. Newer contributions are saved on a higher level. The figure shows how the working memory is changed if new content is introduced into the system that updates the actual knowledge about Entity 4. Since the temporal difference between both working memory states is greater than 500 ms, the validity of Entity 5 is expired and the entity is removed.

addressable by a unique identifier. In SiAM-dp the *Knowledge Manager* is responsible for controlling the access to the LTM.

Working Memory: The Working Memory (WM) holds the entities of the semantic content that is transported within a communicative act. Only resolved content is added to this memory since a unresolved or ambiguous content does not provide additional contributions to the discourse. For every communicative act that contains resolved semantic content, a new entry in the working memory is created. The entries are organised in a list arranged by the time-point of content contribution. Semantic content contributions can manually be annotated with an expiration time by the property *validity*. If the assigned time span is elapsed, the semantic content is removed from the working memory.

Figure 7.10 shows how the WM is updated. The WM state at time point t_0 contains content from four different contributions. *Entity 2* and *Entity 3* have been jointly contributed by one communicative act. *Entity 5* is annotated with a validity of 500 milliseconds. We assume that a new communicative act, which occurs 500 milliseconds or later after time-point t_0, provides new semantic content that extends the knowledge about *Entity 4*. For this, FADE searches the WM for entities that are unifiable with the new entity. If this happens, the old entity is updated with the new content by unification and pushed onto the top level of the memory. Since the validity of *Entity 5* is expired, this entity is removed from the WM.

7.9.2 Reference Resolution

The FADE component uses information from the context for the resolution of referring expressions. In Section 5.5.1 the model for describing references has already been introduced. For each reference type presented there, an individual resolution strategy is applied that involves various parts of the context. Table 7.1 gives an overview of which context is involved in the resolution of the individual reference types.

A special phenomenon that can be handled by FADE is the ellipsis. It must be examined separately since an ellipsis has no referring expression that can be resolved. Moreover, FADE scans the discourse context for previously introduced dialogue acts that contribute semantic content with compatible concepts to the content that is contributed by the ellipsis. If matching instances are found, the FADE component overlays the content of the discourse history with the new instances of the elliptic expression. The resulting semantic content is incorporated into the communicative function of the previous dialogue act.

The overall processing strategy in FADE is organised as depicted in Figure 7.11. For incoming input, the component first looks up for unresolved content and tries to resolve the contributed references. In the next step it checks whether the contribution is annotated as ellipsis and, if this is the case, tries to resolve it. In the final step, new content is propagated to the discourse context manager. If the input contains resolved semantic content entities, these are added to the working memory. The complete input is appended to the discourse history.

Output messages are generated by the dialogue manager which means that no resolution of ambiguous content is necessary. Nevertheless, the concept of the *KnowledgeBaseReference* allows the application designer to reference to content in the environment without knowing the concrete entities. Thus, it is possible to specify an entity with its properties

Table 7.1 – The contexts involved in the individual context resolutions

Reference Type	Scope of application	Involved Context
Deictic Reference	Place Deixis Multimodal Fusion Anaphoric References	Working Memory
Knowledge Base Reference	Reference to world context	Knowledge Base
Spatial Reference	Spatial References	Display Context Knowledge Base
Collection Reference	Collection References	WM
Temporal Reference	Temporal References	World Knowledge
-	Ellipsis	Discourse History

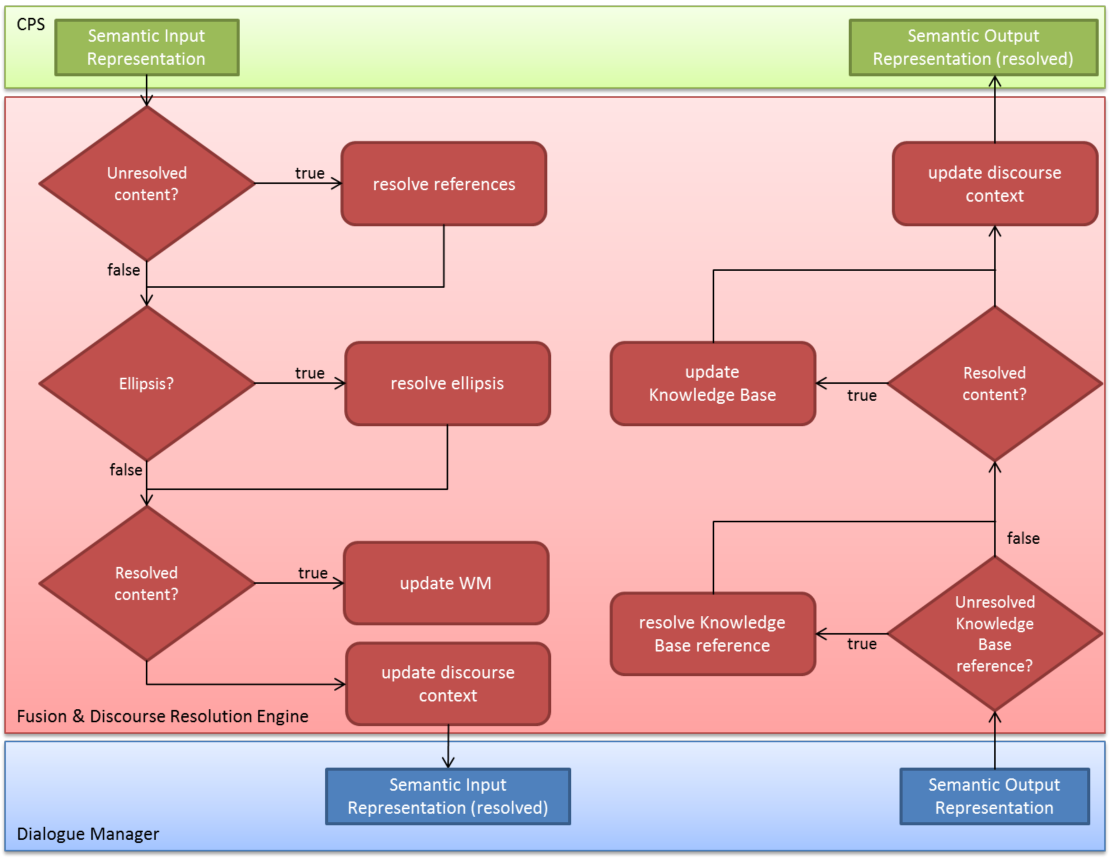

CPS
Semantic Input Representation
Semantic Output Representation (resolved)
Unresolved content?
true
resolve references
false
update discourse context
Ellipsis?
true
resolve ellipsis
false
update Knowledge Base
true
Resolved content?
Resolved content?
true
update WM
false
resolve Knowledge Base reference
true
Unresolved Knowledge Base reference?
update discourse context
Fusion & Discourse Resolution Engine
Semantic Input Representation (resolved)
Semantic Output Representation
Dialogue Manager

Figure 7.11 – The strategy for processing incoming and outgoing semantically represented dialogue acts

in a pattern. The FADE component fulfills the task of finding the concrete entity in the world context. After the resolution of the *KnowledgeBaseReferences*, the discourse context is also updated with the new content before the message is forwarded to the following components.

The process of multimodal integration, mutual disambiguation, and resolution of elliptic and referring expressions is explained in detail with the help of an application example in Section 9.1.1 and 9.1.2.

7.10 Presentation Planning & Distribution

The component that is responsible for output distribution and planning in SiAM-dp receives the semantically represented output from the dialogue manager. Its task is to select the output devices and modalities that are used for presenting the content of the

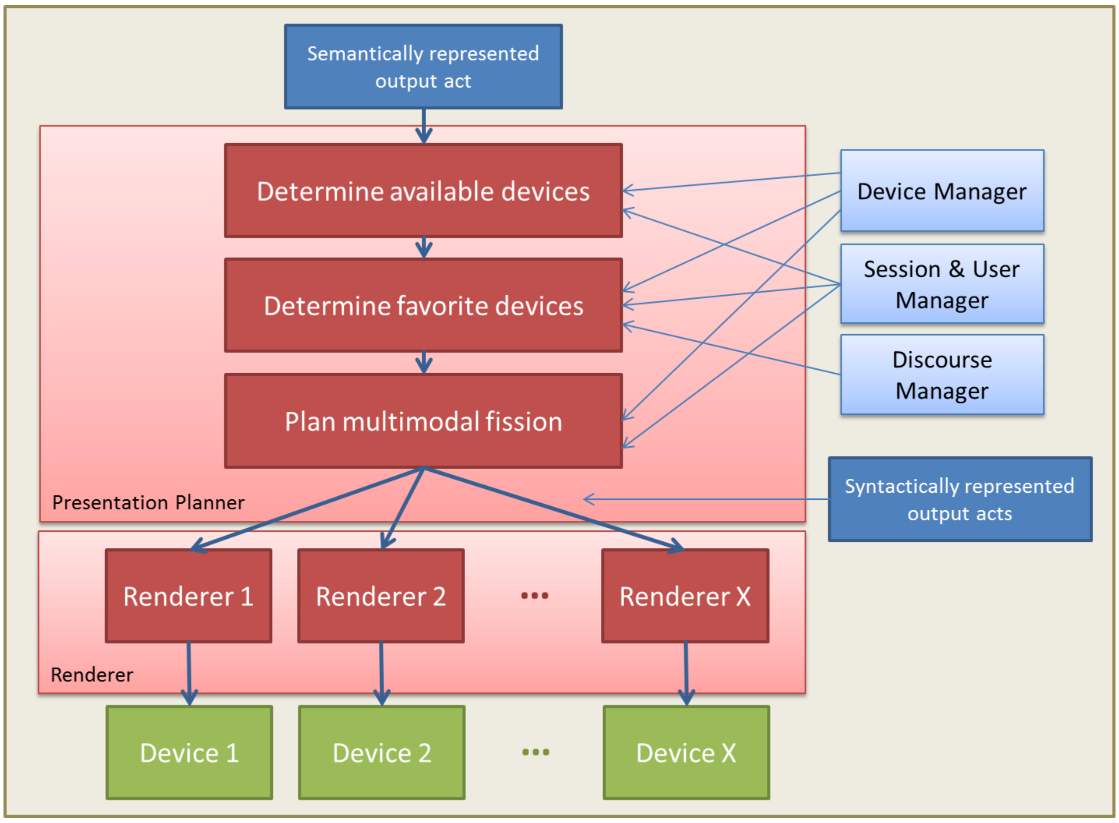


Figure 7.12 – The presentation planning and fission process in SiAM-dp

output act and distribute the data for output to the specific rendering components of the devices.

The processing steps are oriented at the steps introduced in Section 2.2.5. Figure 7.12 shows the adapted fission process in SiAM-dp.

A main difference is that the system first has to determine the available devices in a CPE since, in contrast to the presented approach, system devices in a CPE are distributed throughout the room and can eventually be shared between several users. Also the availability during runtime can change. This is caused by the shutdown of devices, local unavailability if the user is located in the wrong area of the physical environment, or the occupancy of a device by a different user. The assessment of the devices is supported by the device manager that presents information about the availability of devices and the session manager that contains information about the actually used devices and their allocation to the particular users.

The selection of the output modalities is realised in the next step, where the prioritised devices are determined. Here the choice is not only limited to the users' preferences. We discussed in Section 2.2.5 that several context features may play a role for this decision. Thus, the current environment context may be important, for example, in a noisy environment where the use of speech output is not feasible. This, together with information from the user model, like the user's preferences and disabilities, results in a ranked list of devices that should be used for output.

Finally, the multimodal fission is realised. Here, the data for presentation is distributed to the devices in the priority list based on device specific features. This process takes account of the fact that not every device is suitable for presenting every output. In this case, a less ranked device must be selected for the output. Furthermore, some data can be obfuscated, e.g., if it is presented on a public display and violates privacy protection. A further task in this step is to synchronise the presentation of information. Especially for multimodal parallel output with synergistic information and cross-modal references this is quite important, e.g., if a pointing information is combined with speech output. Here, the device manager delivers valuable information about presentation delay of connected devices. The synchronisation is realised by providing additional information about the start time and the duration of an output presentation. A basic requirement therefore is that the connected devices are properly time-stamped and synchronised.

Afterwards, the modality specific output representations are forwarded to the device specific renderer that generate the concrete output presentations on the involved output devices in the CPE.

7.11 Summary

This chapter dealt with the following research question:

5. ***Dialogue System Architecture:*** *Which type of architecture is required for the realisation of distributed coordinated communication in a CPE?*

The chapter described the main architecture of SiAM-dp. This is based on the OSGi framework and basically following the dialogue system reference architecture (Maybury and Wahlster, 1998b). The event management is implemented with the *publish-subscribe pattern* which supports a modular architecture approach and is easily adaptable and extendable. Message filtering is content-based and specified by the pattern model which has been introduced in Section 4.4. It is implemented on the basis of the *OSGi services* functionality.

Four layers group the components of the the SiAM-dp platform: the environment layer, the core layer, the resource, context and knowledge management layer, and the backend layer. Principally components can be distinguished between those that are part of the main data flow of an interaction and those that are orthogonally accessible for arbitrary components and responsible for management tasks and the allocation of information.

Finally, the following individual components of the core SiAM-dp deployment were described in detail:

Components for the main processing workflow:

- Dialogue Manager
- Fusion & Discourse Resolution
- Presentation Planning & Distribution
- Speech Recognition Interpreter
- GUI Event Interpreter

Supplemental Components:

- Project Manager
- Session & User Manager
- Device Manager
- Knowledge Manager
- GUI Manager
- Speech Grammar Manager

8 Development Tools

The previous chapter introduced the runtime environment of the Situation Adaptive Multimodal Dialogue Platform (SiAM-dp), which is responsible for the practical realisation of a dialogue application. The description of the content and the behaviour of the application is based on declarative models as described in Chapter 6. These models must be be defined by the application engineer. The SiAM-dp Software Development Kit (SDK), which is presented in this chapter, supports his work and contains a collection of tools and editors for the creation of new domain-specific dialogue applications. Furthermore, it provides a wizard for the extension of the dialogue platform with new devices and thus allows to easily integrate new modalities into the platform, which can be reused in other dialogue applications. The toolkit is fully integrated into the Eclipse *Rich Client Platform (RCP)*, which is shortly introduced in the first section. Other tools and views can be used for monitoring and debugging running dialogue applications.

8.1 The Eclipse Rich Client Platform

Eclipse is an open-source programming tool and provides an integrated development environment (IDE) for various programming languages. It has been originally developed by IBM for the programming language Java. In 2004 IBM founded the Eclipse Consortium[1] which now is primarily responsible for the further development of Eclipse.

Eclipse offers the possibility to extend the IDE with new extensions and thus to adapt the environment to custom demands. Especially for this, Eclipse contains the integrated plugin development environment (PDE) (Clayberg and Rubel, 2008). In the terminology of Eclipse the expression *plugin* stands for a software component that provides additional content to the environment. Since Eclipse is built upon the runtime environment Equinox OSGi, a plugin is in technical terms an OSGi bundle that is started together with the

[1]https://eclipse.org/

IDE. By using the so-called *Extension Points* of Eclipse it can provide additional content to the workbench or extend the functionality of other already available plug-ins.

The basic RCP framework consists only of the basic application framework, including action sets, perspectives, views, and editors. The tooling for IDE specific aspects is optional and can be installed on demand. This includes , e.g., functionalities for source editing, refactoring, compiling, debugging or building. Furthermore, the PDE allows to bundle several plugins to features and deploy them via a web-based update manager that hosts new content on so-called *p2 repositories*.

8.2 SiAM Workbench

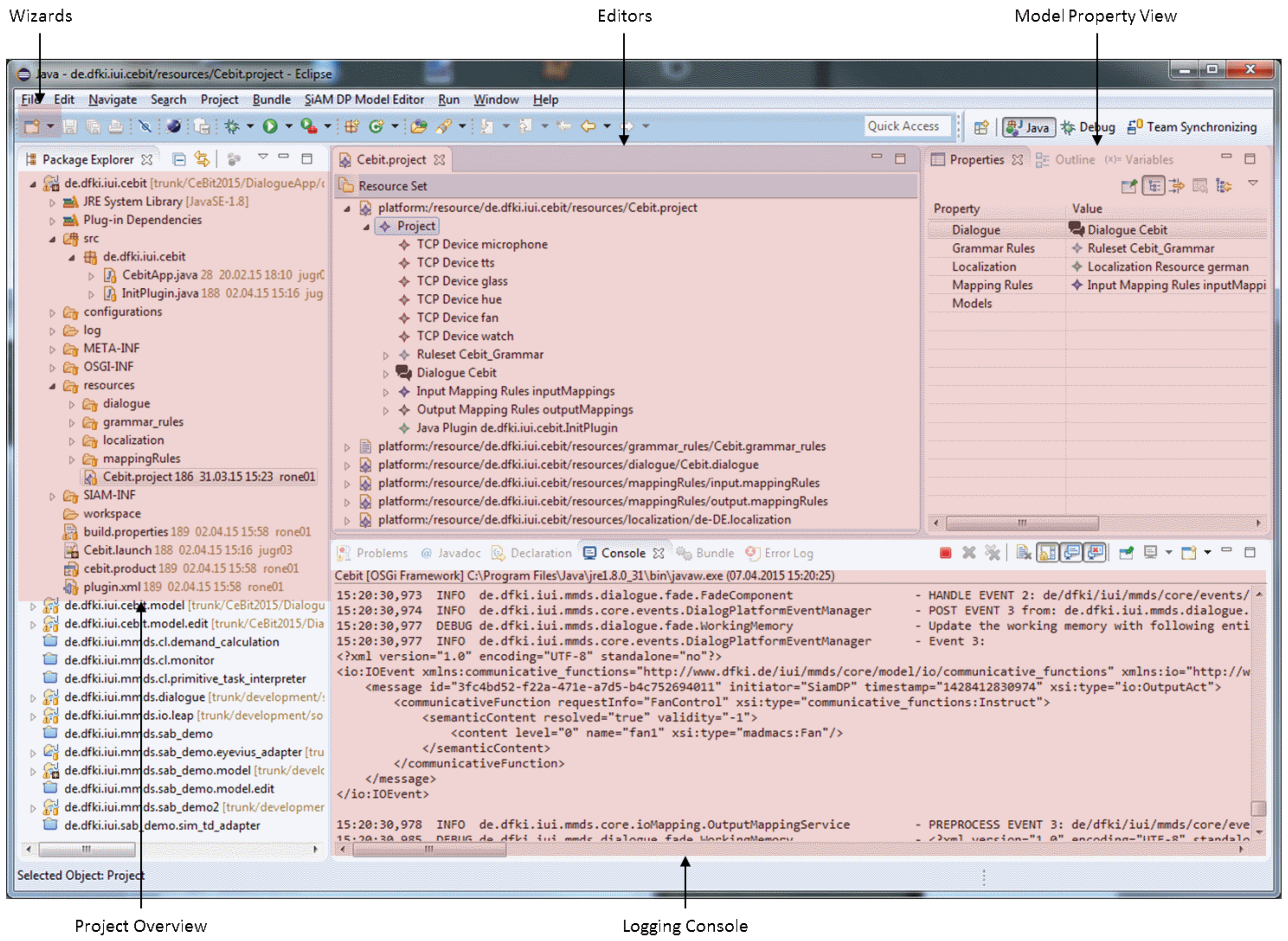
An important factor for the acceptance and success of a new development platform is the set of available tools that enable developers an easy entry into the creation of new applications. We aim to achieve this with the SiAM-dp workbench, a kit of tools that supports developers during the complete development process. It contains three wizards for the creation of new model instances, new application projects, and new projects for the integration of new devices.

Figure 8.1 gives an overview of the SiAM-dp workbench. It extends the Eclipse RCP concepts with an own set of actions and views. In the following sections these components are described in detail.

8.2.1 Wizard for New Applications

The most of the first steps, when creating a new dialogue application from scratch, can be automated. This includes the generation of Java code, the correct configuration of the OSGi plugin, launch configurations, and first templates for model instances that actually form the real content of an application.

A project generator is responsible for creating the relevant Java classes and configuration files and putting them at the right place in the file hierarchy. The generation process is based on templates filled with meta-information that must be manually provided by the developer. For the structured input of this meta-information we define a new Eclipse extension point, a wizard. This PDE concept collects the required information from the developer with the help of a sequence of modal dialogue windows (see Figure 8.2). The user can switch between the single dialogue windows and provide the necessary information. The wizard validates the information and only creates the new project if all necessary information is available. After the developer has confirmed the given information with the *Finish* button, the project generator creates the skeletal structure of the new project. Basis of this project is a standard plugin project which is extended with some SiAM-dp relevant code and configurations. For this, the generator fills templates



de.dfki.iui.cebit [trunk/CeBit2015/DialogueApp/
JRE System Library [JavaSE-1.8]
Plug-in Dependencies
src
de.dfki.iui.cebit
CebitApp.java 28 20.02.15 18:10 jugr0
InitPlugin.java 188 02.04.15 15:16 jug
configurations
log
META-INF
OSGI-INF
resources
dialogue
grammar_rules
localization
mappingRules
Cebit.project 186 31.03.15 15:23 rone01
SIAM-INF
workspace
build.properties 189 02.04.15 15:58 rone01
Cebit.launch 188 02.04.15 15:16 jugr03
cebit.product 189 02.04.15 15:58 rone01
plugin.xml 189 02.04.15 15:58 rone01
de.dfki.iui.cebit.model [trunk/CeBit2015/Dialogu
de.dfki.iui.cebit.model.edit [trunk/CeBit2015/Dia
de.dfki.iui.mmds.cl.demand_calculation
de.dfki.iui.mmds.cl.monitor
de.dfki.iui.mmds.cl.primitive_task_interpreter
de.dfki.iui.mmds.dialogue [trunk/development/
de.dfki.iui.mmds.io.leap [trunk/development/so
de.dfki.iui.mmds.sab_demo
de.dfki.iui.mmds.sab_demo.eyevius_adapter [tru
de.dfki.iui.mmds.sab_demo.model [trunk/develo
de.dfki.iui.mmds.sab_demo.model.edit
de.dfki.iui.mmds.sab_demo2 [trunk/developmer
de.dfki.iui.sab_demo.sim_td_adapter

Resource Set
platform:/resource/de.dfki.iui.cebit/resources/Cebit.project
Project
TCP Device microphone
TCP Device tts
TCP Device glass
TCP Device hue
TCP Device fan
TCP Device watch
Ruleset Cebit_Grammar
Dialogue Cebit
Input Mapping Rules inputMappings
Output Mapping Rules outputMappings
Java Plugin de.dfki.iui.cebit.InitPlugin
platform:/resource/de.dfki.iui.cebit/resources/grammar_rules/Cebit.grammar_rules
platform:/resource/de.dfki.iui.cebit/resources/dialogue/Cebit.dialogue
platform:/resource/de.dfki.iui.cebit/resources/mappingRules/input.mappingRules
platform:/resource/de.dfki.iui.cebit/resources/mappingRules/output.mappingRules
platform:/resource/de.dfki.iui.cebit/resources/localization/de-DE.localization

Property	Value
Dialogue	Dialogue Cebit
Grammar Rules	Ruleset Cebit_Grammar
Localization	Localization Resource german
Mapping Rules	Input Mapping Rules inputMappi
Models	

```
Cebit [OSGi Framework] C:\Program Files\Java\jre1.8.0_31\bin\javaw.exe (07.04.2015 15:20:25)
15:20:30,973  INFO  de.dfki.iui.mmds.dialogue.fade.FadeComponent               - HANDLE EVENT 2: de/dfki/iui/mmds/core/events/
15:20:30,974  INFO  de.dfki.iui.mmds.core.events.DialogPlatformEventManager    - POST EVENT 3 from: de.dfki.iui.mmds.dialogue.
15:20:30,977  DEBUG de.dfki.iui.mmds.dialogue.fade.WorkingMemory               - Update the working memory with following enti
15:20:30,977  INFO  de.dfki.iui.mmds.core.events.DialogPlatformEventManager    - Event 3:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<io:IOEvent xmlns:communicative_functions="http://www.dfki.de/iui/mmds/core/model/io/communicative_functions" xmlns:io="http://w
  <message id="3fc4bd52-f22a-471e-a7d5-b4c752694011" initiator="SiamDP" timestamp="1428412830974" xsi:type="io:OutputAct">
    <communicativeFunction requestInfo="FanControl" xsi:type="communicative_functions:Instruct">
      <semanticContent resolved="true" validity="-1">
        <content level="0" name="fan1" xsi:type="madmacs:Fan"/>
      </semanticContent>
    </communicativeFunction>
  </message>
</io:IOEvent>

15:20:30,978  INFO  de.dfki.iui.mmds.core.ioMapping.OutputMappingService       - PREPROCESS EVENT 3: de/dfki/iui/mmds/core/eve
```

Figure 8.1 – Overview of the in Eclipse RCP integrated SiAM-dp workbench

of the relevant project files with the information collected from the developer. The following resources are part of the generated project:

Java Code: Every dialogue application contains a class that is derived from the `AbstractDialogueApplication` class. In this class the file name of the model for the project definition (see Section 6.4) is provided. This class can be automatically generated by the project generator.

Runtime Configurations: Component configurations are collected in one central directory and distributed with the OSGi configuration manager. The configuration directory is permanently monitored for file changes during runtime, which are directly forwarded to the affected plug-ins. The project generator creates basic configurations for some SiAM-dp internal components like the logger or the FADE component.

OSGi component description: The component description provides an entry point to the application class and defines relevant dependencies to internal SiAM-dp services. In the bundle's manifest the bundle configuration is defined. This includes the bundle name, bundle version, and dependencies to other bundles of the OSGi framework and SiAM-dp.

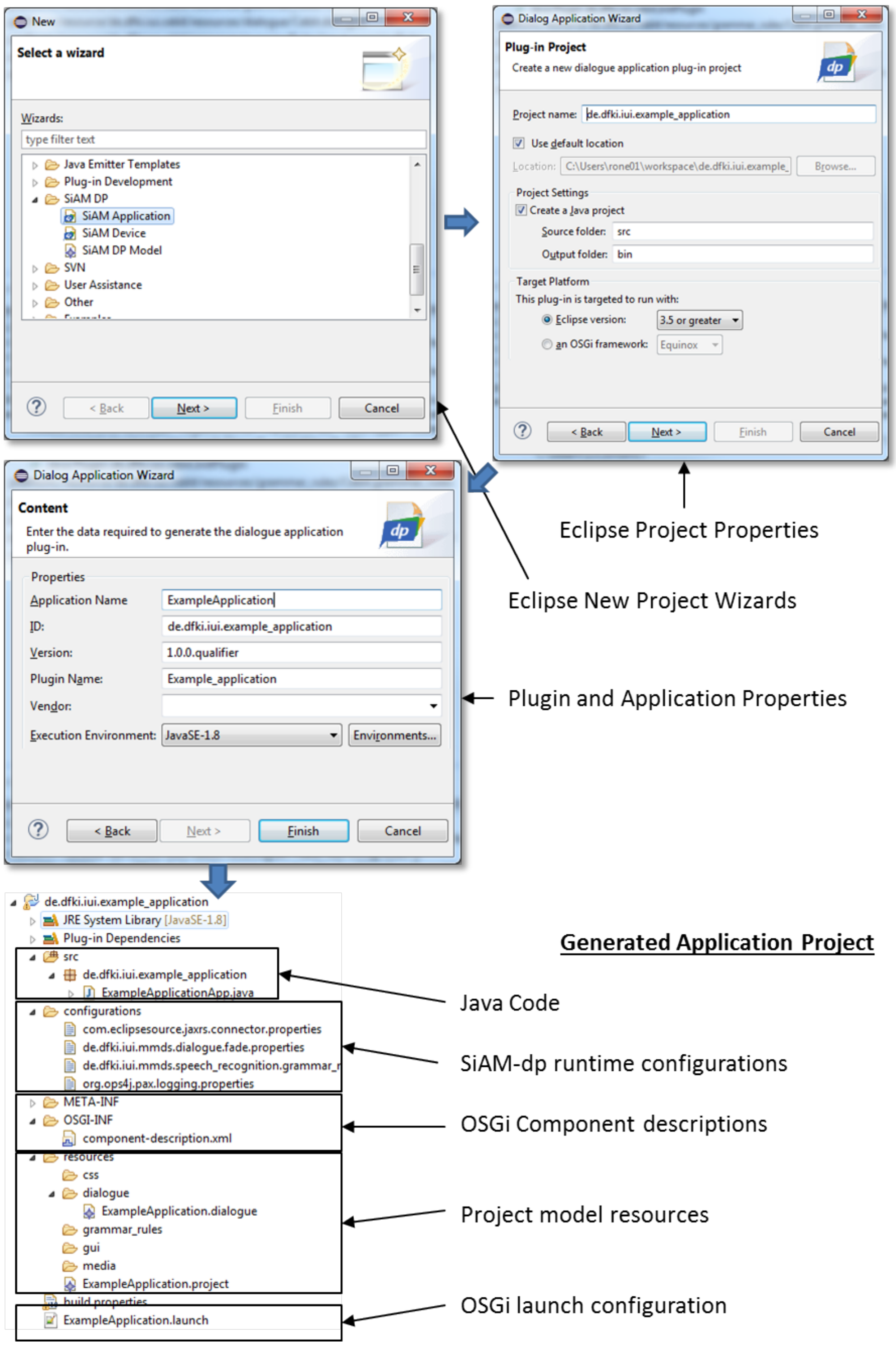
Project model resources: The wizard creates a folder structure for resources of the dialogue application. This includes models for the project, Graphical User Interfaces (GUIs), and grammar rules, as well as other resource like media files and style cheats. The most directories must be filled with content by the developer but the resource folder already contains a project definition model with a reference to an empty dialogue specification model.

Launch configuration: Since the dialogue application runs within a OSGi framework, it is important to specify a correct launch configuration. There exist some dependencies to bundles of the SiAM-dp runtime but also to OSGi and external support bundles. Some of the bundles must also be started in the right order. Finally, the correct Java virtual machine arguments and system properties must be set.

8.2.2 New Device Wizard

Supporting the rapid extension with new devices is one of the main goals of the workbench. Beside the possibility to connect a device via diverse protocols (XML with a TCP connection, Thrift with a TCP connection, XML via REST), a developer can also create an OSGi bundle that implements the connection to the a device API.

Projects that implement a new device are called *Device Projects*. Similar to the *"New Application Wizard"* the workbench also contains a wizard for the creation of a new device project. The approach here is similar to the one for new applications. Within


Eclipse Project Properties
Eclipse New Project Wizards
Plugin and Application Properties
Generated Application Project
Java Code
SiAM-dp runtime configurations
OSGi Component descriptions
Project model resources
OSGi launch configuration

New

Select a wizard

Wizards:

type filter text

- Java Emitter Templates
- Plug-in Development
- SiAM DP
  - SiAM Application
  - SiAM Device
  - SiAM DP Model
- SVN
- User Assistance
- Other

< Back | Next > | Finish | Cancel

Dialog Application Wizard

Plug-in Project

Create a new dialogue application plug-in project

Project name: de.dfki.iui.example_application

[✓] Use default location

Location: C:\Users\rone01\workspace\de.dfki.iui.example_ | Browse...

Project Settings

[✓] Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

(●) Eclipse version: 3.5 or greater

( ) an OSGi framework: Equinox

< Back | Next > | Finish | Cancel

Dialog Application Wizard

Content

Enter the data required to generate the dialogue application plug-in.

Properties

Application Name	ExampleApplication
ID:	de.dfki.iui.example_application
Version:	1.0.0.qualifier
Plugin Name:	Example_application
Vendor:	
Execution Environment:	JavaSE-1.8

Environments...

< Back | Next > | Finish | Cancel

- de.dfki.iui.example_application
  - JRE System Library [JavaSE-1.8]
  - Plug-in Dependencies
  - src
    - de.dfki.iui.example_application
      - ExampleApplicationApp.java
  - configurations
    - com.eclipsesource.jaxrs.connector.properties
    - de.dfki.iui.mmds.dialogue.fade.properties
    - de.dfki.iui.mmds.speech_recognition.grammar_r
    - org.ops4j.pax.logging.properties
  - META-INF
  - OSGI-INF
    - component-description.xml
  - resources
    - css
    - dialogue
      - ExampleApplication.dialogue
    - grammar_rules
    - gui
    - media
    - ExampleApplication.project
  - build.properties
  - ExampleApplication.launch

Figure 8.2 – Overview of the modal windows in the application wizard

some modal dialogue windows, the developer has to provide the necessary information for the new project like the device name and some bundle settings. Afterwards, a project generator creates a new plugin project with the relevant files (Figure 8.3):

> **Java Code:** The main class in a device project represents a device component and is derived from the class `AbstractDeviceComponent`. This class already implements the communication with the event manager and provides abstract method stubs for handling output messages with the device as target.
>
> **OSGi configurations:** In the component description the relevant dependencies to the services for device management and event management are specified. Furthermore, a bundle manifest is generated.

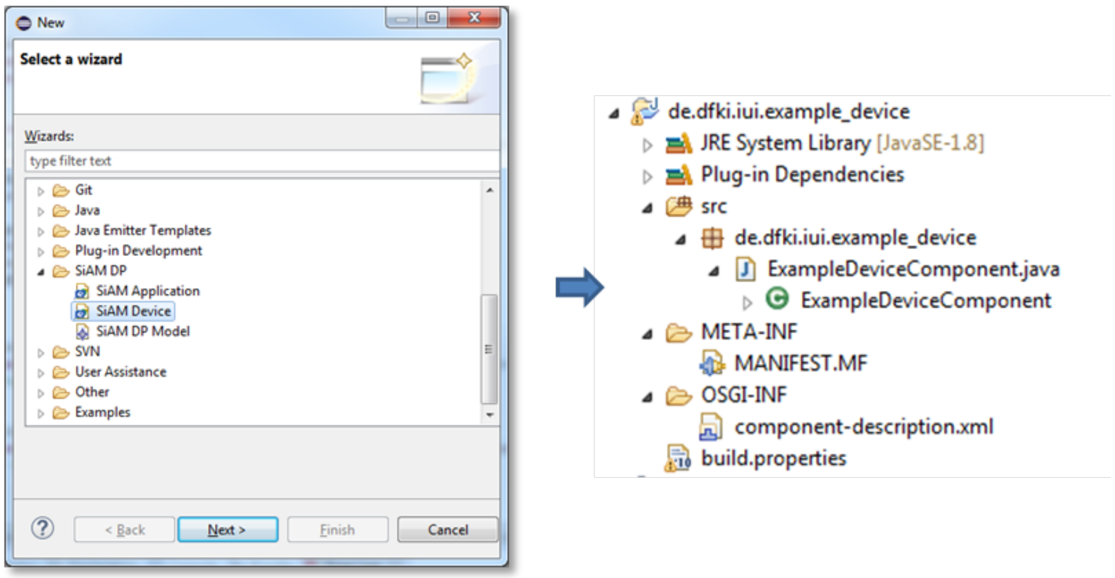


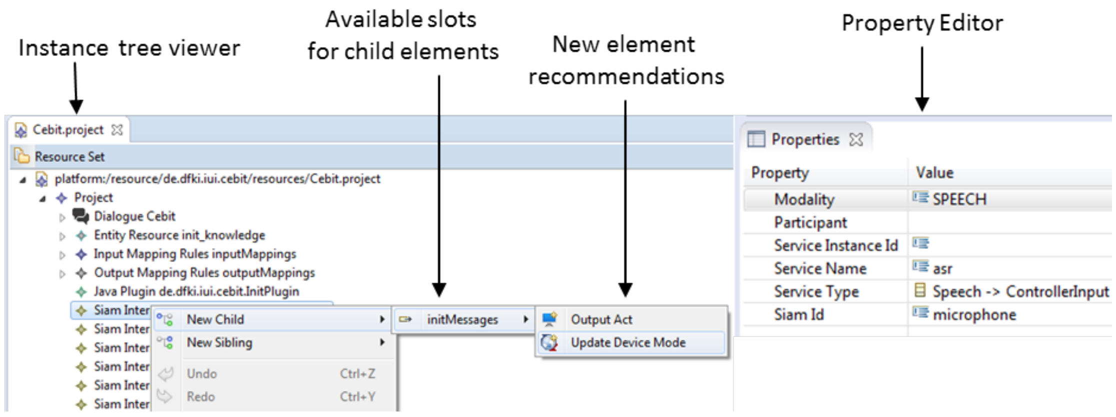
Figure 8.3 – Overview of the modal windows in the device wizard

8.2.3 Extended EMF instance editor

The EMF framework allows to generate code for tree based standard EMF model editors that are integrated in the Eclipse platform. This editor consists of two views as depicted in Figure 8.4. The tree view based editor gives an overview of the instance being edited and its containing elements. In this view a developer can add new child instances or remove existing ones. When adding new child elements, the editor makes suggestions for the type of the new element based on the underlying metamodel of the instance. Attributes of the instance can be edited in an own property view. For the SiAM-dp workbench the generated editor has been extended with features that increase the functionality and usability:

Global clipboard: The copy & paste functionality for model instances has been extended to a global clipboard which allows to copy instances between different editor windows and edited files.

Dynamic model extension: In the standard version of the editor, recommendations for new elements only consider concepts of a static set of already existing models that cannot be modified after the deployment of the workbench. Since SiAM-dp allows to introduce new domain-specific models, we adapted this approach to work on a dynamic set of models that can be modified later on by the application developer. Furthermore, the proposed elements are now categorized by the slot they are attached to and thus provide a more structured menu.


Instance tree viewer
Available slots for child elements
New element recommendations
Property Editor
Cebit.project
Resource Set
platform:/resource/de.dfki.iui.cebit/resources/Cebit.project
Project
Dialogue Cebit
Entity Resource init_knowledge
Input Mapping Rules inputMappings
Output Mapping Rules outputMappings
Java Plugin de.dfki.iui.cebit.InitPlugin
Siam Inter
Siam Inter
Siam Inter
Siam Inter
Siam Inter
Siam Inter
New Child
New Sibling
Undo Ctrl+Z
Redo Ctrl+Y
initMessages
Output Act
Update Device Mode
Properties

Property	Value
Modality	SPEECH
Participant	
Service Instance Id	
Service Name	asr
Service Type	Speech -> ControllerInput
Siam Id	microphone

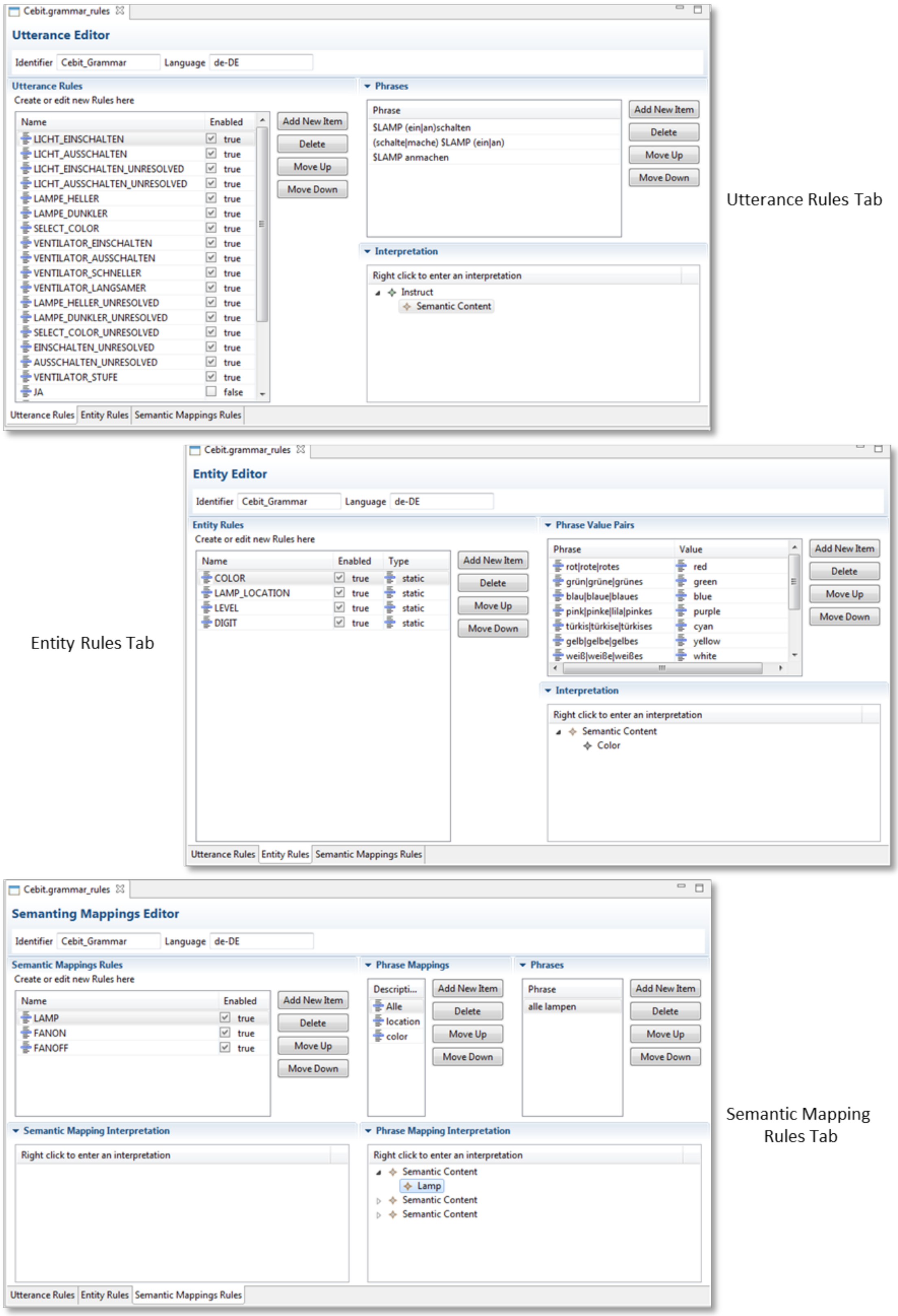
Figure 8.4 – The views of the extended EMF instance editor

8.2.4 Grammar Rule Editor

The previously introduced tree view editor is able to handle every model deployed with the SiAM-dp platform. However, for some models it makes sense to provide editors that are tailored to a specific model type in order to support a rapid development approach. The speech grammar rule editor (Figure 8.5) is one of these editors. Here, individual content of the grammar rules can be edited in particular subviews in a way that is most suitable for presentation and modification.

The editor is divided into three tabs, one specific tab for every grammar rule type. The different grammar rule types are explained in detail in Section 6.2. Global settings like the identifier for the rule set and the target language are available in each tab.

Utterance Rules Tab:
This tab shows a list view containing each utterance rule in the rule set. In the list view it is possible to add and remove rules, edit their identifier, and the *enabled* status for the rule at application start. If one of the rules is selected, it is possible to edit the specified phrases for the rule in a separate list view. This view additionally



Utterance Editor

Identifier Cebit_Grammar Language de-DE

Utterance Rules
Create or edit new Rules here

Name	Enabled
LICHT_EINSCHALTEN	✓ true
LICHT_AUSSCHALTEN	✓ true
LICHT_EINSCHALTEN_UNRESOLVED	✓ true
LICHT_AUSSCHALTEN_UNRESOLVED	✓ true
LAMPE_HELLER	✓ true
LAMPE_DUNKLER	✓ true
SELECT_COLOR	✓ true
VENTILATOR_EINSCHALTEN	✓ true
VENTILATOR_AUSSCHALTEN	✓ true
VENTILATOR_SCHNELLER	✓ true
VENTILATOR_LANGSAMER	✓ true
LAMPE_HELLER_UNRESOLVED	✓ true
LAMPE_DUNKLER_UNRESOLVED	✓ true
SELECT_COLOR_UNRESOLVED	✓ true
EINSCHALTEN_UNRESOLVED	✓ true
AUSSCHALTEN_UNRESOLVED	✓ true
VENTILATOR_STUFE	✓ true
JA	false

Phrases

Phrase
$LAMP (ein|an)schalten
(schalte|mache) $LAMP (ein|an)
$LAMP anmachen

Interpretation
Right click to enter an interpretation
Instruct
Semantic Content

Entity Editor

Identifier Cebit_Grammar Language de-DE

Entity Rules
Create or edit new Rules here

Name	Enabled	Type
COLOR	✓ true	static
LAMP_LOCATION	✓ true	static
LEVEL	✓ true	static
DIGIT	✓ true	static

Phrase Value Pairs

Phrase	Value
rot|rote|rotes	red
grün|grüne|grünes	green
blau|blaue|blaues	blue
pink|pinke|lila|pinkes	purple
türkis|türkise|türkises	cyan
gelb|gelbe|gelbes	yellow
weiß|weiße|weißes	white

Interpretation
Right click to enter an interpretation
Semantic Content
Color

Semanting Mappings Editor

Identifier Cebit_Grammar Language de-DE

Semantic Mappings Rules
Create or edit new Rules here

Name	Enabled
LAMP	✓ true
FANON	✓ true
FANOFF	✓ true

Phrase Mappings

Descripti...
Alle
location
color

Phrases

Phrase
alle lampen

Semantic Mapping Interpretation
Right click to enter an interpretation

Phrase Mapping Interpretation
Right click to enter an interpretation
Semantic Content
Lamp
Semantic Content
Semantic Content

Figure 8.5 – The three tabs of the grammar rule editor

verifies the ABNF syntax correctness of the given phrases. The annotations for the semantic interpretation of the rule can be specified in a separate tree view.

Entity Rules Tab:
This tab contains a list view with all entity rules of the grammar rule set. It allows to specify the identifier for this rule and to set the *enabled* and *type* attribute. The mappings from ABNF phrases to values are defined in a separate list view. Every entity rule can have a *SemanticContent* instance as interpretation result, which is defined in a separate tree view.

Semantic Mapping Rules Tab:
Semantic mapping rules allow to directly map phrases to *SemanticContent* instances. This tab contains a list view with all semantic mapping rules in the grammar rule set that allows to set the identifier and the *enabled* attribute. The various phrase mappings of a selected rule are presented in a separate list view. If one of these mappings is selected, it is possible to edit the assigned phrases in a third list view and the definition of the semantic interpretation in a tree view.

8.2.5 Graphical Dialogue Model Editor

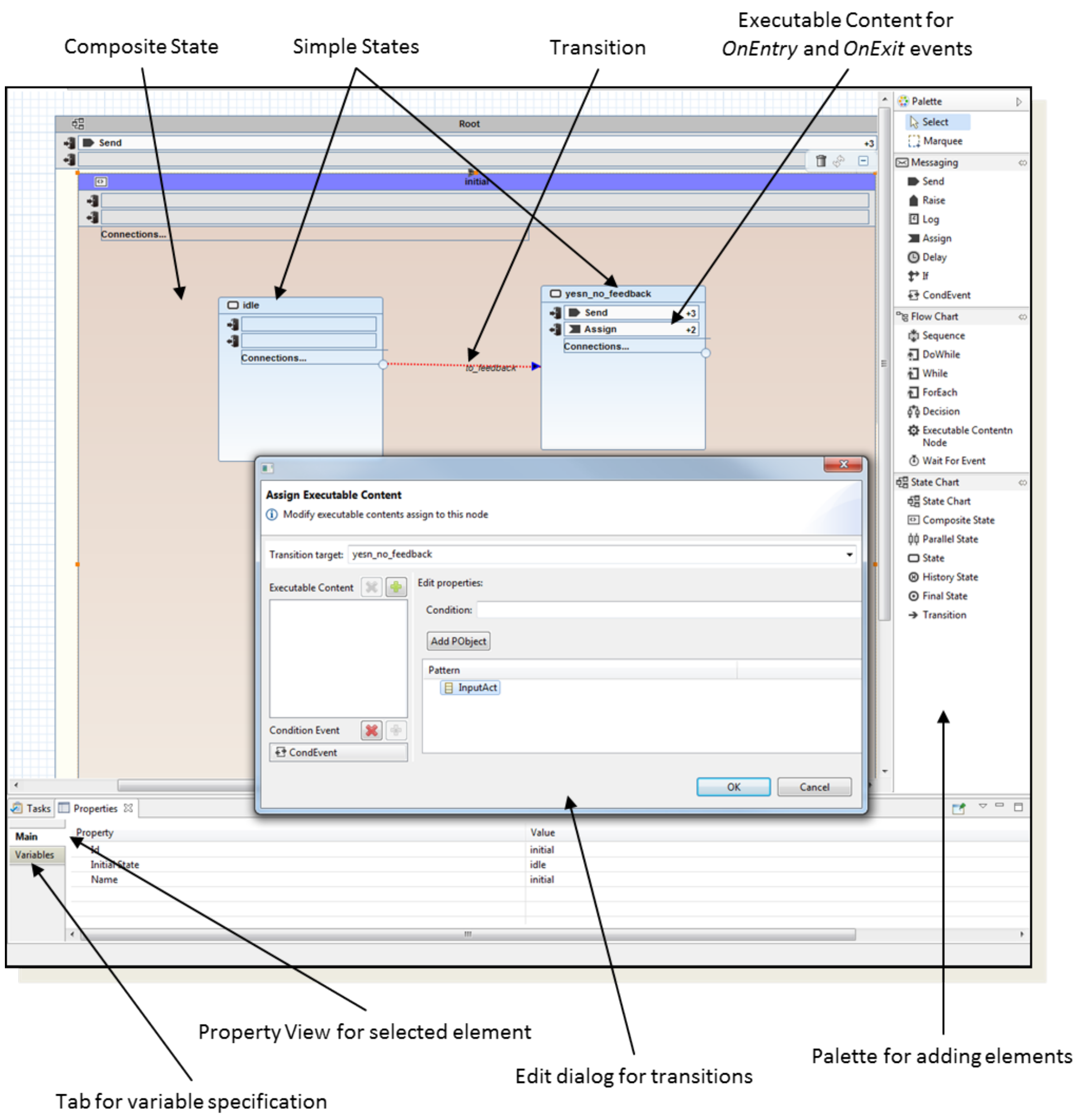
The SiAM-dp dialogue model editor (Figure 8.6) is a visual designer for the graphical definition of dialogue models (see Section 6.1). Since the main content of a dialogue model consists of flow charts and statecharts, it has a great practical value for the developer to visualise the program flow using a graph view.

The editor has been developed with Graphiti, a graphical tooling infrastructure that enables the rapid development and customisation of state-of-the-art diagram editors for EMF models[2]. Graphiti is Eclipse-based and can directly integrate editors into the Eclipse RCP workbench.

The main window of the dialogue editor is a canvas that shows elements of the models and their relations to each other. Thus, the hierarchical structure of *composite states* and their inner *simple states* are immediately apparent to the developer. *Transitions* between states can be defined by drawing an array from the source to the target state. Single elements of the canvas can be moved per drag and drop. A tool palette at the right side of the window allows the user to select a new element and to add it to the model.

Furthermore, several editor views are included that support the modification of the various model instances' properties. Here, we emphasised that the edit functionalities are appropriate to the content that is modified. For example, tree editors are used for more complex and hierarchical content (e.g., other embedded model instances) and list based property views for simple attributes.

[2]https://eclipse.org/graphiti/



Transition target: yesn_no_feedback

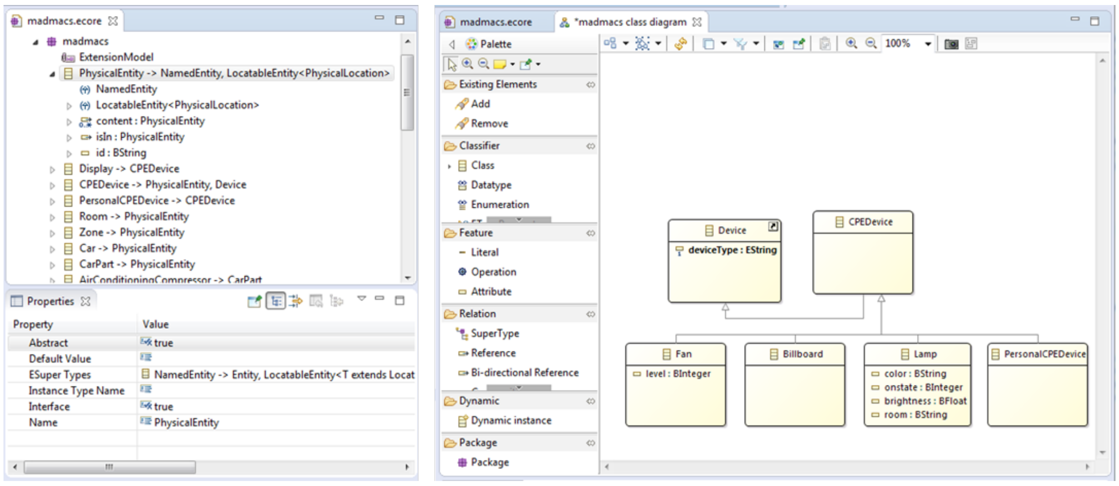
Property	Value
Id	initial
Initial State	idle
Name	initial

Figure 8.6 – The graphical editor for dialogue specification models

8.2.6 Domain Ontology Editor

The domain ontologies in SiAM-dp are modelled with the meta-modelling language EMF. This framework already provides Eclipse-based standard model editors, which can be used by the application developer to create and modify application specific domain ontologies. These editors are included into the SiAM-dp workbench and accessible via the *Modelling Perspective*. The standard editor for EMF models is a tree-based editor that hierarchically lists all concepts and their containing attributes and references. Alternatively, the framework contributes a graphical editor that presents models in a class diagram view (see Figure 8.7).

Furthermore, the EMF framework contains a comprehensive tooling for EMF models. A model generator can be used by the developer in order to generate Java classes that cover the defined ontology concepts. Another functionality of this generator is to deploy content providers for the concepts in the ontology. Their purpose is to fill the previously described instance editor with the necessary information in order to display model instances of the domain-specific ontology. Thus, it is possible to directly create and modify instances of the new domain.



```
madmacs
  ExtensionModel
  PhysicalEntity -> NamedEntity, LocatableEntity<PhysicalLocation>
    NamedEntity
    LocatableEntity<PhysicalLocation>
    content : PhysicalEntity
    isIn : PhysicalEntity
    id : BString
  Display -> CPEDevice
  CPEDevice -> PhysicalEntity, Device
  PersonalCPEDevice -> CPEDevice
  Room -> PhysicalEntity
  Zone -> PhysicalEntity
  Car -> PhysicalEntity
  CarPart -> PhysicalEntity
  AirConditioningCompressor -> CarPart
```

Property	Value
Abstract	true
Default Value	
ESuper Types	NamedEntity -> Entity, LocatableEntity<T extends Locat
Instance Type Name	
Interface	true
Name	PhysicalEntity

Figure 8.7 – Editors for modelling the domain ontology

8.3 Runtime Tools

In the previous section we presented those elements of the SiAM-dp toolkit that support the development of massively multimodal dialogue applications at design time. The following set of tools provides assistance during the runtime of an application.

8.3.1 Application Debug GUI

Especially when debugging an application, it is a large benefit if the developer can monitor the behaviour of the runtime environment in detail and directly inspect states and variables of the application. Beside the Eclipse Java Development Tools (JDT) that are integrated in the Eclipse Platform by default, we provide a separate GUI that can be started together with a dialogue application and provides detailed information and control functions for monitoring and debugging issues. A screenshot of the GUI is presented in Figure 8.8.

State-machine log
An important part of the information in this GUI is about the internally running statechart engine. It contains a log view that lists every state change, which is triggered during runtime. If a new event is fired that matches the condition of an active transition, the engine changes from the active state to the target state that is specified by the transition. Thus, the target state becomes the new active state. If a state change occurs, the monitor logs the ID of the fired transitions and the states that have been left and entered. For the identification of states, the list provides two identifiers. The first is the dialogue node ID, which is the ID of the associated node as specified in the dialogue model. Since the dialogue model is converted to SCXML before it is loaded into the statechart engine, the node structure and the real identifiers can sometimes differ. So, the real internal state ID is additionally presented in this view.

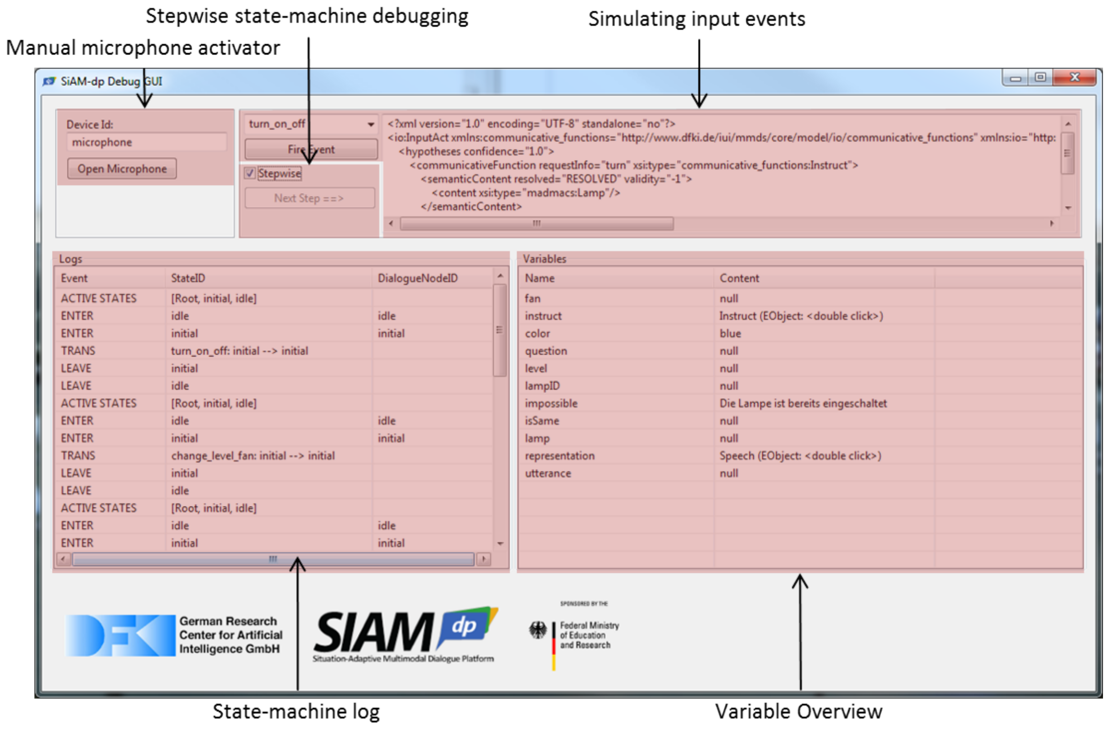
Variable Overview
This overview lists all variables in the actual scope of the active states and their values. More detailed information about EMF instances can be retrieved by clicking on the concrete value. In this case, an extra window opens that shows the content of the EMF instance in a serialised XML representation.

Stepwise state-machine debugging
Sometimes several transitions of the statechart engine fire successively. In this case, the tracking of variable changes is difficult for an application developer. With this option, transitions are triggered only if manually initiated by the developer.

Manual microphone activator
This button opens the microphone of the connected speech recognition device with the given ID. Since speech recognition is very often used in multimodal applications, this button is a shortcut for triggering an *open microphone* event.

Manual input event trigger
Sometimes it is difficult to trigger input events for testing, due to a non-finished concrete device implementation or the development of an application at a working place beyond the target domain with the appropriate infrastructure. This function allows to simulate the messages that are sent by the input device. For this, the transition

patterns in the dialogue model are collected and appropriate least matching messages proposed. The developer can select the messages, modify them in serialised XML form, and send them to the event manager of the running application.


Stepwise state-machine debugging
Simulating input events
Manual microphone activator
SiAM-dp Debug GUI
Device Id:
microphone
Open Microphone
turn_on_off
Fire Event
Stepwise
Next Step ==>
German Research Center for Artificial Intelligence GmbH
SIAM dp
Situation-Adaptive Multimodal Dialogue Platform
Federal Ministry of Education and Research
State-machine log
Variable Overview

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<io:InputAct xmlns:communicative_functions="http://www.dfki.de/iui/mmds/core/model/io/communicative_functions" xmlns:io="http:
  <hypotheses confidence="1.0">
    <communicativeFunction requestInfo="turn" xsi:type="communicative_functions:Instruct">
      <semanticContent resolved="RESOLVED" validity="-1">
        <content xsi:type="madmacs:Lamp"/>
      </semanticContent>
```

Logs

Event	StateID	DialogueNodeID
ACTIVE STATES	[Root, initial, idle]	
ENTER	idle	idle
ENTER	initial	initial
TRANS	turn_on_off: initial --> initial	
LEAVE	initial	
LEAVE	idle	
ACTIVE STATES	[Root, initial, idle]	
ENTER	idle	idle
ENTER	initial	initial
TRANS	change_level_fan: initial --> initial	
LEAVE	initial	
LEAVE	idle	
ACTIVE STATES	[Root, initial, idle]	
ENTER	idle	idle
ENTER	initial	initial

Variables

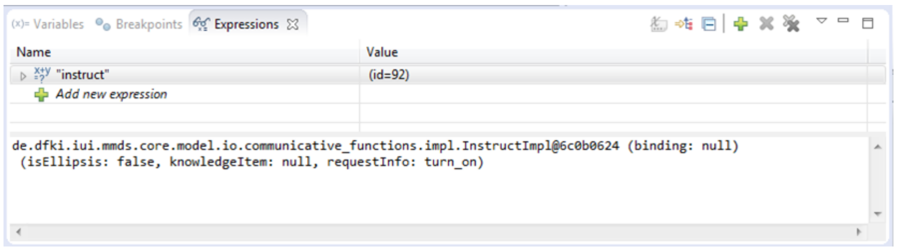
Name	Content
fan	null
instruct	Instruct (EObject: <double click>)
color	blue
question	null
level	null
lampID	null
impossible	Die Lampe ist bereits eingeschaltet
isSame	null
lamp	null
representation	Speech (EObject: <double click>)
utterance	null

Figure 8.8 – The SiAM-dp debug GUI

8.3.2 JDT Extensions

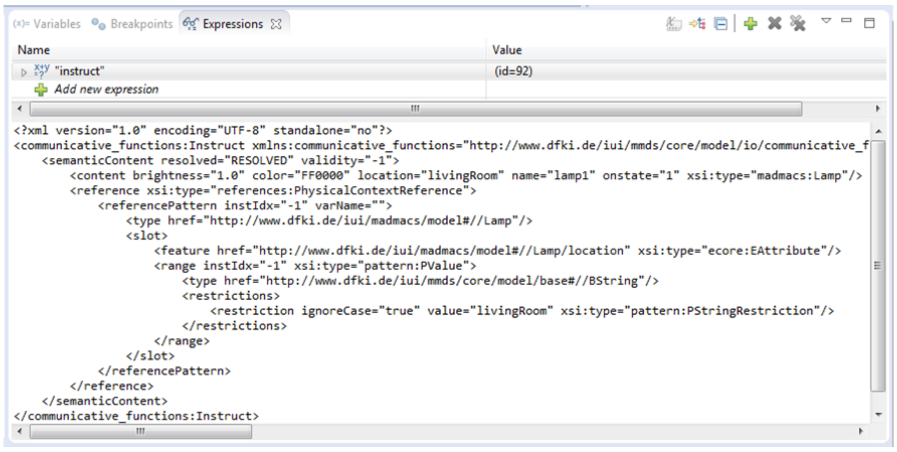
The Eclipse Java development tools (JDT) provide some plug-ins that support the Java development in Eclipse, including debugging and IDE user interfaces. One of the functionalities is a debug view that displays the result of a Java expression during runtime. It allows developers to specify so-called 'detail formatters' that customise the representation of Java instances. In the default case, the `toString()` method of the object is used. In the case of EMF instances, this method only returns a short summary of the object type and its attributes (see Figure 8.9a). The containing objects are hidden in this representation.

For developing and debugging dialogue applications, it is helpful to have a quick and complete overview of the content of an instance. For this issue, the SiAM-dp platform provides a specific detail formatter that displays the EMF instance in serialised XML from (compare Figure 8.9b). Thus, the content is more human readable allowing the developer a better understanding of the running workflows during a debug session.



```
de.dfki.iui.mmds.core.model.io.communicative_functions.impl.InstructImpl@6c0b0624 (binding: null)
 (isEllipsis: false, knowledgeItem: null, requestInfo: turn_on)
```

(a) Standard JDT detail formatter for EMF objects.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<communicative_functions:Instruct xmlns:communicative_functions="http://www.dfki.de/iui/mmds/core/model/io/communicative_f
    <semanticContent resolved="RESOLVED" validity="-1">
        <content brightness="1.0" color="FF0000" location="livingRoom" name="lamp1" onstate="1" xsi:type="madmacs:Lamp"/>
        <reference xsi:type="references:PhysicalContextReference">
            <referencePattern instIdx="-1" varName="">
                <type href="http://www.dfki.de/iui/madmacs/model#//Lamp"/>
                <slot>
                    <feature href="http://www.dfki.de/iui/madmacs/model#//Lamp/location" xsi:type="ecore:EAttribute"/>
                    <range instIdx="-1" xsi:type="pattern:PValue">
                        <type href="http://www.dfki.de/iui/mmds/core/model/base#//BString"/>
                        <restrictions>
                            <restriction ignoreCase="true" value="livingRoom" xsi:type="pattern:PStringRestriction"/>
                        </restrictions>
                    </range>
                </slot>
            </referencePattern>
        </reference>
    </semanticContent>
</communicative_functions:Instruct>
```

(b) Extended JDT detail formatter for EMF objects.

Figure 8.9 – Comparison of the standard and extended representation of EMF objects

8.3.3 Automatic GUI prototyping with HTML 5

SiAM-dp defines a model for the abstract specification of GUIs (see Section 6.3). With this model the application developer can define the structure and content of graphical elements. The design aspects for the presentation can be expressed by attaching additional style sheets to the document, a list of stylistic rules. The concrete realisation of the graphical output is implemented by graphical renderers.

Experiences from previous work on multimodal dialogue applications in the Theseus project (Wahlster et al., 2014) showed that the development of dialogue applications and graphical renderer not always make simultaneous progress. This often led to the situation that application developers designed a graphical output presentation based on the GUI model, but were not directly able to test the system during development since the graphical renderer had not been finished.

To overcome this problem, SiAM-dp provides the bundle *de.dfki.iui.mmds.io.html* that is responsible for two tasks. First, it contains a full-fledged web-socket server that establishes bidirectional connections to web browser clients. Second, it contains a generic GUI renderer that dynamically creates HTML 5 content based on the actual display context of the GUI devices in the dialogue application. Figure 8.10 shows the generated presentation of a GUI model instance in the web browser. Here, developers can directly preview the result of their graphical output. On the other way around, user interactions with the presented HTML page are sent back to the dialogue platform event manager, after they are converted to a format that is compliant to the IO model of SiAM-dp. Thus, the complete interaction of the user with a GUI can be simulated.

Based on this approach for GUI previews, the web-socket functionality has more and more been refined and is now at a state where it can serve as an adequate output device. Here, a main focus has been placed on the customisation of the generated HTML code. The generated HTML pages fully support style sheets that are attached to the GUI model. Cascading Style Sheets (CSSs) enable designers to create user interfaces on a professional level by adding styles like fonts, colours and spacing, or easily define and exchange the common style of a complete application.

Furthermore, the developer can inject Java script or register arbitrary methods that manipulate the generated HTML document before it is deployed to the browser. All these functionalities offer a large degree of freedom for generating a concrete realisation of GUIs .

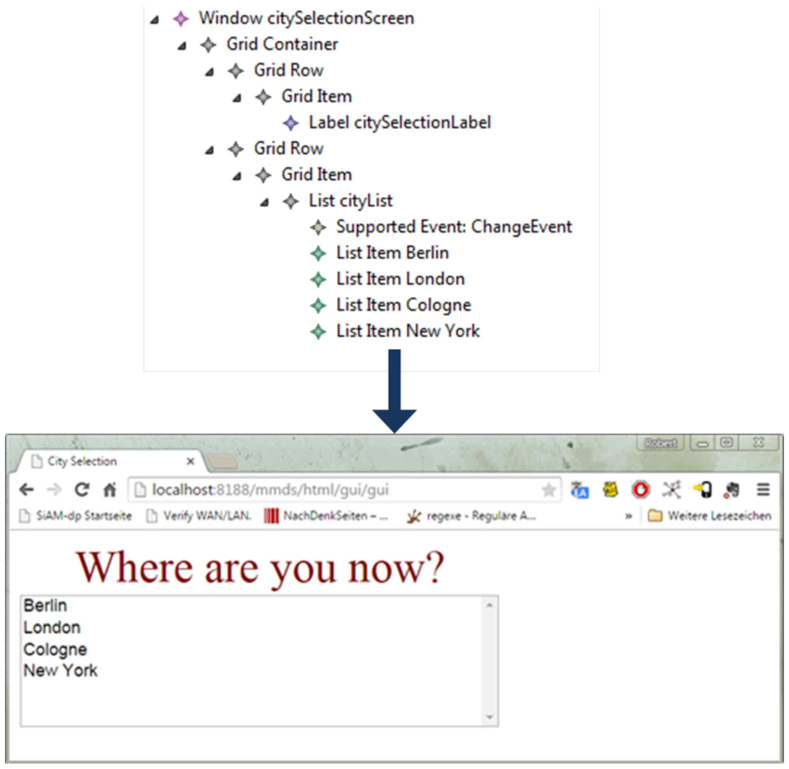
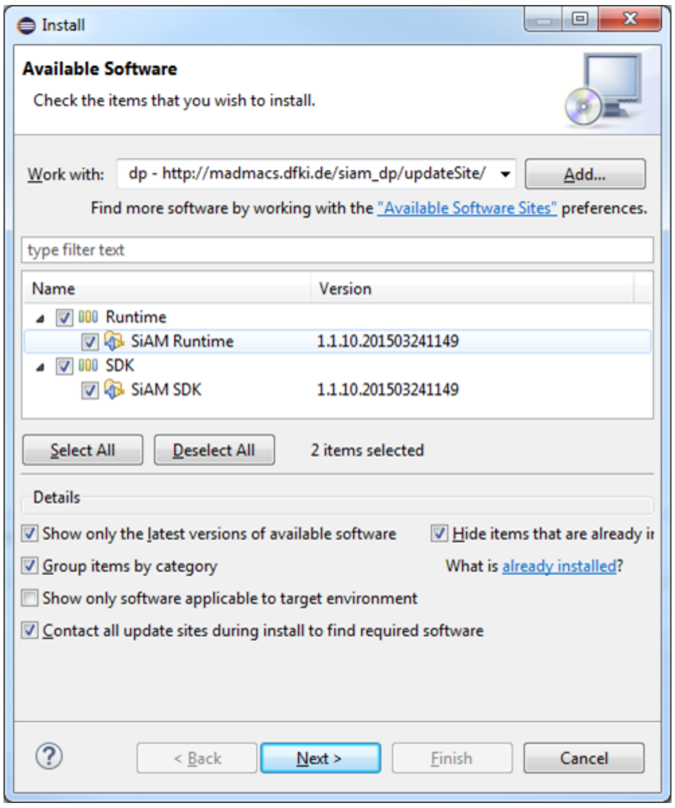


Figure 8.10 – Example for a generated HTML page from a GUI model instance. The model defines a grid layout with two grid rows that both contain one element. The first element is a label, the second element a list with city names. The generated HTML page consists of a label entry and a combo box that contains the items of the list.

8.4 SiAM-dp deployment

SiAM-dp is deployed via an Eclipse p2 repository and an update site. An update site is a special website for hosting eclipse features and plugins and giving additional information like product descriptions and copyright information. The Eclipse Update Manager can read the sites' manifest files and automatically install and update plugins that are deployed on the repository.

The SiAM-dp plugins are distributed by two features (Figure 8.11): first, the plugins that constitute the core components of the SiAM-dp runtime environment. Second, the plugins that extend the eclipse workbench with SiAM-dp development tools. The plugins of this feature contain the wizards, editors, and JDT extensions, which have been introduced in this chapter.



Install

Available Software

Check the items that you wish to install.

Work with: dp - http://madmacs.dfki.de/siam_dp/updateSite/ Add...

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☑ Runtime	
☑ SiAM Runtime	1.1.10.201503241149
☑ SDK	
☑ SiAM SDK	1.1.10.201503241149

Select All Deselect All 2 items selected

Details

☑ Show only the latest versions of available software
☑ Hide items that are already i
☑ Group items by category
What is already installed?
☐ Show only software applicable to target environment
☑ Contact all update sites during install to find required software

< Back Next > Finish Cancel

Figure 8.11 – The Eclipse Update Manager with the SiAM-dp update site as target location

8.5 Summary

This chapter dealt with the following research question:

6. ***Tool Support:*** *How is an integrated development environment designed that simplifies and accelerates the creation of multimodal dialogue applications?*

This chapter described the application development tools of SiAM-dp that support developers in the rapid design of new dialogue applications. The tools are directly integrated into the Eclipse RCP environment. In summary, the developers are supported at three stages of the development process:

Project Generation
Wizards support the developer in the generation of new projects and create project templates as starting point for the implementation. The workbench contains the following wizards:

- New Application Wizard
- New Device Wizard
- New Model Instance Wizard

Model Declaration
Editors that are specialized for concrete models and a general editor for all SiAM-dp models. The workbench contains the following editors:

- General SiAM-dp model editor
- Grammar rule editor
- Graphical dialogue model editor

Debugging and Testing
These tools support the developer in testing and debugging a running dialogue application. They allow the developer to monitor and stepwise control the dialogue workflow and give previews on the appearance of graphical output. The following features are presented:

- Extensions to the Eclipse Java Development Tools (JDT)
- Application debug GUI for monitoring and controlling the dialogue workflow
- HTML 5 preview of graphical output

9 Applications

For the evaluation of the practical applicability and acceptance, it is a fundamental step to use the dialogue platform for the development of concrete massively multimodal dialogue applications. In this chapter some projects are presented whose demonstrator applications have been developed on the basis of SiAM-dp. The concrete applications presented in the following section are each assigned to their corresponding projects.

The first three demonstrators are explained in more detail. They are part of the SiAM project in which context SiAM-dp has been developed, tested, and refined with the experience that could be collected during the development process. Furthermore, they are an excellent representative example to explain the strategies and processes for multimodal fusion and reference resolution and therefore supplement the section about these topics (Section 7.9).

The remaining sections give an overview of research projects in which other working groups of the DFKI are involved that highly benefit from the development strategies deployed with SiAM-dp. The structure of the sections in each case is as follows: First, a main overview about the projects and their goals is given. The following parts show in detail which devices are involved in the applications. In order to highlight the massively multimodal approach of SiAM-dp, each demonstrator is summarised in a table that lists all devices and communication channels that have been integrated. Here, each device is classified by the attributes that have been identified in Section 5.2.

9.1 SiAM Project Demonstrator

SiAM-dp was developed in the SiAM (Situation-Adaptive Multimodal Interaction for Innovative Mobility Concepts of the Future) project and constitutes the central in-car platform of the project. The final result of the project is a system for smart vehicles that incorporates multimodal dialogue interaction with an approach for assessing the driver's focus of attention. In the demo scenario of the project, the knowledge from many sources, inside and outside the car, is combined to provide the optimal type of interaction for the current situation.

In the following subsections, four scenarios of the demonstrator are presented which illustrate the individual facets of the platform's strengths in a concrete application. All of these applications have been demonstrated live at various occasions, i.a., the project's review by the scientific advisory board of the DFKI, and the DFKI-TechDay (Mitrevska et al., 2015).

9.1.1 Multimodal Control of Car Functions

In this scenario the support of multimodal interaction in the SiAM dialogue platform is demonstrated. It comprises the multimodal control of several in-car actuators like the windows, outside mirrors, or the turn signals. The main idea of the adaptive approach is that the various input modalities can synergistically or alternatively be employed, depending on the actual situation and the preferences of the driver. Table 9.1 shows the classification of the devices that are integrated in this scenario.

In-Car Setup

Figure 9.1 shows the setup of the input devices involved in this scenario. An overview of the component architecture is given in Figure 9.2. In total, three input modalities have been integrated into the dialogue application:

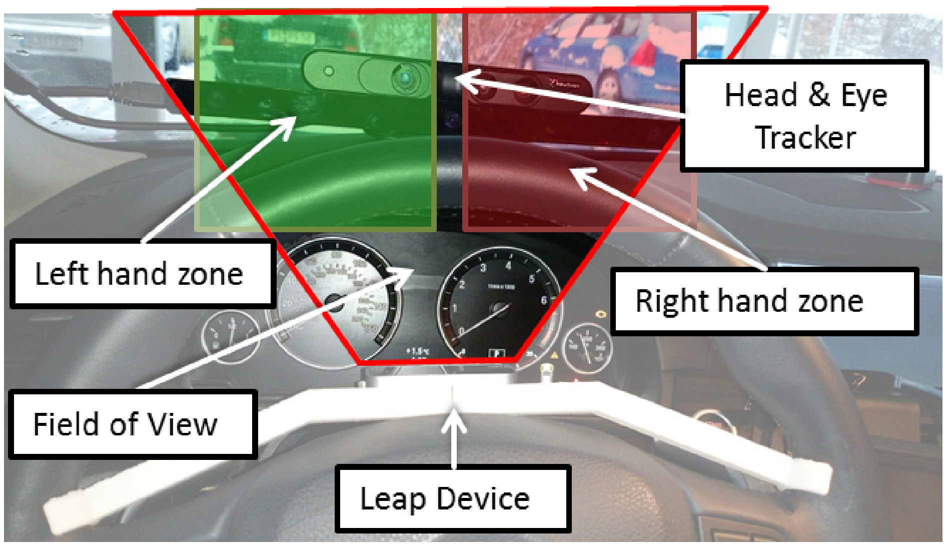


Figure 9.1 – Input devices in the scenario for car function control

Device	Service	Type	Modality	Device Position	Interaction Range	User number
Eye Tracker and Head tracker	EyeVIUS	Controller	visual	static	near-field	single
Leap Motion	micro gestures	Controller	visual	static	near-field	single
Microphone	speech recognition	Controller	auditory	static	near-field	single
Car Controls	CanBUS	Actuator	-	static	-	-

Table 9.1 – Classification of the devices integrated in the car control scenario

EyeVIUS is a research prototype that provides third party applications the possibility to integrate information about the driver's attention in their existing platforms (Moniri and Müller, 2014). It consists of three different modules to monitor the driver's eye-gaze, head-pose, and facial expression. They collect and process data from a head tracker and an eye tracker that have been placed in front of the driver behind the steering wheel. This information is evaluated together with a model of the car environment and provides assumptions about objects in the driver's focus to the dialogue application.

Micro Gestures are directly performed with the hands on the steering wheel. For the recognition of finger movements, a Leap Motion Controller[1] is placed at the center of the steering wheel. We have designed a special holder for the device and

[1]https://www.leapmotion.com

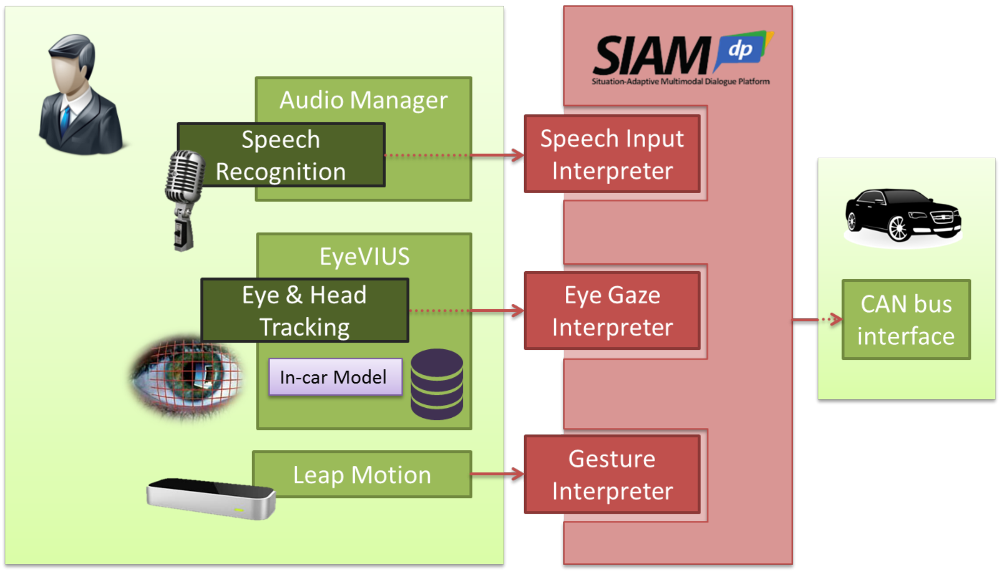


Figure 9.2 – The external components connected to SiAM-dp in the multimodal control scenario

printed it with a 3D printer. The Leap Motion controller contains two monochromatic IR cameras and three infrared LEDs. The LEDs generate a 3D pattern of IR light dots. If fingers are placed within a roughly hemispherical area above the device, they reflect the infrared signal which is registered by the IR camera and afterwards processed by a Leap Motion Software for the creation of a hand model.

Figure 9.1 shows that only a narrow part of the steering wheel lies within the highlighted field of view of the leap device. Hence, for this prototype the hands must be placed within this region. Furthermore, the steering wheel obfuscates the thumbs and large parts of the hands. This handicaps the Leap Motion software in generating a complete hand model, making if difficult to allocate detected fingers to the left or right hand. We face this issue by defining separate zones for the left and the right hand and assigning detected fingers to the hand of the corresponding zone. Thus, we can recognise the number of stretched fingers for every hand (unfortunately not the type of finger due to the missing hand skeleton model). The zones and the coordinates of the detected fingers are processed by a gesture interpretation component that generates a semantic interpretation of the driver's finger gestures. The interpretation results are based on some micro gesture strategies we have defined for the demonstrator. They are explained in detail in the next section.

Speech Input is a common hands-free input modality for the interaction in the car. For this, the dialogue application establishes a TCP connection to the *SiAM-dp Audio Manager*, a standalone application for the easy and robust access to audio hardware and audio services. It supports the IO protocol of the SiAM-dp model and can directly be connected to the dialogue application. The most common supported speech services are ASR (Automatic Speech Recognition) and TTS (Text-to-Speech synthesis). In this scenario, the Audio Manager accesses the Microsoft Speech API (SAPI) which is an integral part of any actual Windows distribution. SAPI provides a grammar-based speech recognition interface and a speech synthesis component.

The car functions are accessed via an interface with the car's CAN bus (Controller Area Network). The CAN bus is a vehicle bus standard that allows micro controllers and devices to communicate without a host computer. It is widespread and can be found in nearly every modern car. Besides diagnostic issues, it is used for the control of car actuators. We access this interface for the control of the in-car functions via a standalone application that is connected to the dialogue application via a TCP connection.

Interaction strategy

Since we support multimodal interaction in this scenario where different parts of a user's intention originate from different input devices, we identified two types of contributions that input can provide. The first is the context or the concrete actuator instance the driver intends to control, e.g., the front right window or the left turn signal. The second

Table 9.2 – Interaction concepts for the gesture only control

Finger Level: / Actuator	Lower	Zero	Upper
Turn Signals	Flash Right	Deactivate	Flash Left
Front Windows	Open	Stop	Close
Outside Mirrors	Fold	-	Unfold

is the function the driver wants to perform, like opening (a window) or folding (the outside mirrors).

Our interaction strategy allows that both input types can be contributed to by a single input device but also as a multimodal combination of two devices. We support the following combinations:

Speech Only Both context and function can be communicated in one utterance spoken by the driver. Examples for possible utterances are *"Open the right windows"* or *"Fold the outside mirrors in"*. Furthermore, content can subsequently be contributed. E.g., the first utterance *"the right front window"* introduces the actuator instance and the following contribution *"open it"*, which contains an anaphoric reference to the previous utterance, specifies the function.

Gestures Only We have developed an interaction concept that assigns each type of contribution to one hand. The left hand specifies the context, more precisely the actuator to control. One stretched finger indicates the turn signals, two fingers the front windows, and three fingers the outside mirrors. The right hand specifies the function. Two fingers of the hand are stretched to a plain level zero position (see Figure 9.3). Relative to this plain level, we additionally define an upper and a lower position. The concrete function is dependent on the actuator (see Table 9.2).

Speech & Gesture In this multimodal combination each content type is contributed by an individual modality. The dialogue platform is responsible for the fusion of this input. One possibility is that the actuator is introduced by a left hand micro gesture and the function is given by speech. E.g., one stretched finger of the left hand combined with the utterance *"open"* opens the front windows. The other possibility is to introduce the actuator by speech and to specify the function by right hand motions. In this case, the utterance *"the right front window"* combined

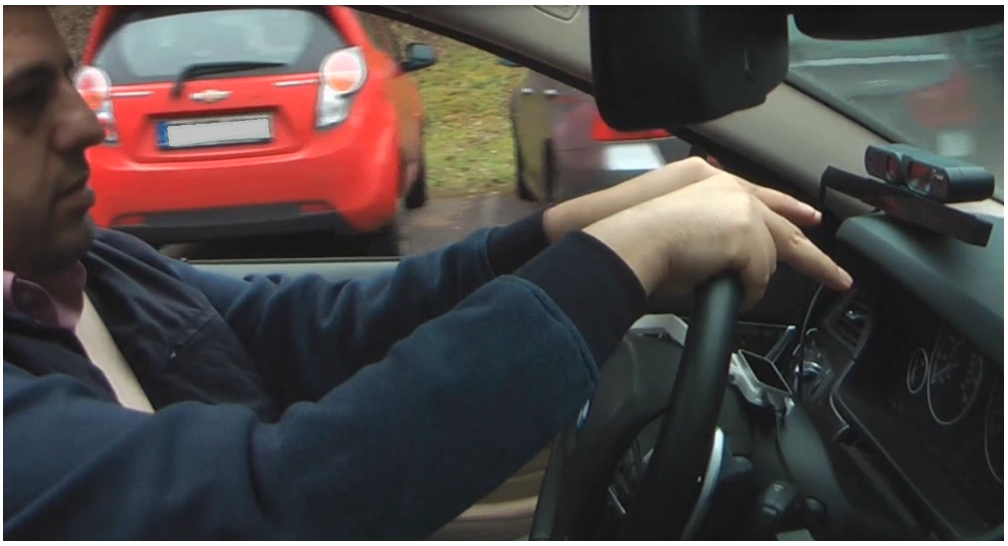


Figure 9.3 – Micro-Gesture Control in the car

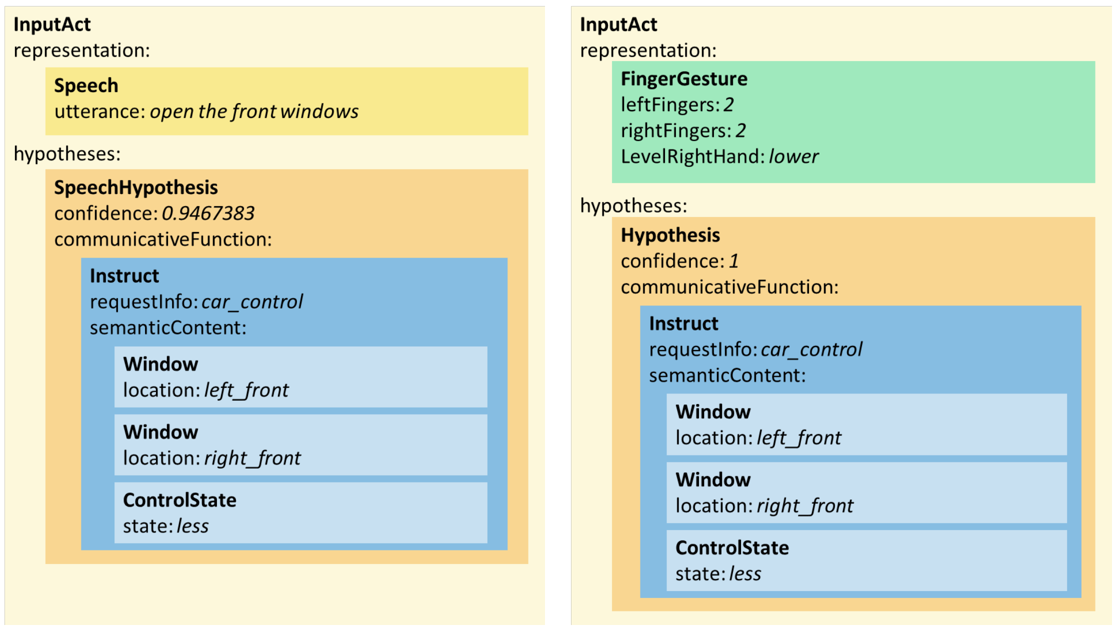
with the right hand's fingers on a lower level opens the right front window.

Speech & Focus of Attention Gaze is a fundamental deictic marker (Meurant, 2008). However, the stationary eye tracker, which is used to monitor the driver's gaze direction, has a limited functioning angle. Thus, we also use the head pose of the user as an indicator of the focus of attention if the eyes are not in the opening angle of the eye-tracker (when the head is turned to the right or left). The communicative meaning of an eye-gaze or a head pose, for an interaction, is comparable to a deictic gesture. Nevertheless, we always have to bear in mind that the gaze (or a head pose) upon an object can occur unconsciously. Because of this, we use this information only to complement an incompletely spoken utterance that solely specifies the function but not the actuator to control. A gaze (or a head pose) alone can not trigger an action. Thus, we avoid unwanted behaviour.

It is possible that more than one actuator is located in the line of sight, e.g., if the driver looks through a window at the right outside mirror. Here the **mutual disambiguation** of speech and eye gaze helps to resolve the ambiguity of the semantics behind a spoken utterance, like *"fold this mirror in"*, because it is possible to fold in an outside mirror but not a window.

Representation and fusion of the driver's intentions

In SiAM-dp we developed models for the description of the communicative function of a user's input and the implicitly transported semantic content. The representation is based on TFS. On this higher semantic level, the input description is device independent



(a) Speech Input **(b)** Micro gesture input

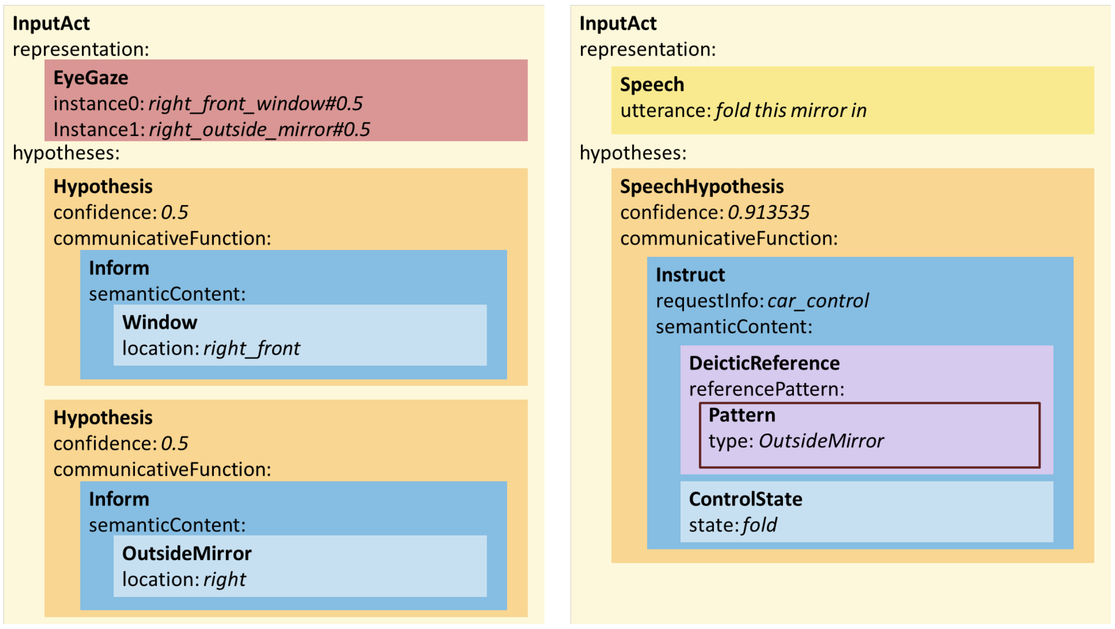
Figure 9.4 – Examples for the representation of unimodal input

and can be used for modality fusion as described in this section.

Unimodal

Figure 9.4a shows the (simplified) input message representation of the speech utterance *"open the front windows"* after it has passed the speech input interpreter. The content of the slot *representation* is the syntactic representation of the input act. In this example, it contains the recognised utterance (other meta-information is not presented for better clarity). The speech input interpreter shifts the content to a semantic level and extracts two relevant pieces of information from this utterance. The first is the context which is presented in the form of both addressed actuators, the left and the right front window. The second is the control function with the desired control state *less*, which stands for opening the windows. Both bits of information are added as the semantic content to the representation of the communicative function of the input act, an instruction to control a function of the car.

The second example (Figure 9.4b) shows the same command generated by a gesture only interaction. In this case, the driver stretches two fingers of the left hand and holds the right hand's fingers on a lower level. The syntactic representation of the input act now contains micro gesture specific information. This is information about the number of stretched fingers for every hand and the level of the right hand's fingers. The gesture interpreter generates an input interpretation based on the above mentioned interaction strategies. The resulting content is the same as in the speech input act with the same



(a) EyeGaze **(b)** Speech input

Figure 9.5 – Examples for the representation of multimodal input

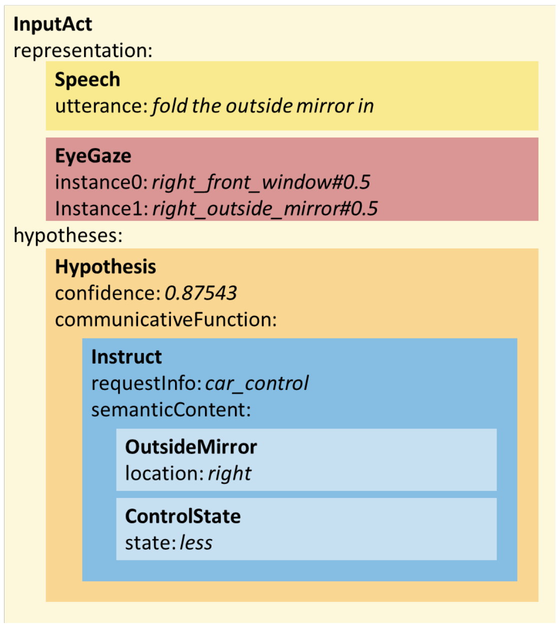
intention.

The reaction of the system is determined only based on the content of the communicative function and is thus completely device independent. Especially in a more complex and adaptive system, this is a relevant advantage for maintainability and extensibility.

Multimodal

This example examines a multimodal input case with the combination of eye-gaze and speech input. The driver is looking at the right outside mirror and gives the command: *"Fold this mirror"*. In this utterance it is not clear which of the two mirrors is referenced, so the system has to consider the additional information of the eye-gaze interpreter.

Figure 9.5a shows an input example for the EyeVIUS component. The syntactic representation provides the identifier of two actuators with an attached confidence value, both coded together in a string. Here, two entities are provided since the driver is looking through the window at the mirror, which is also a possible actuator. The eye-gaze interpreter takes this input and raises it on a semantic level. Thus, we obtain two hypotheses, one for each possible entity. The communicative function behind the input act is interpreted as an *Inform*, the introduction of a new entity to the system, either for the right window or for the right outside mirror. Both hypotheses adopt the confidence value of the syntactic representation.



```
InputAct
representation:
    Speech
    utterance: fold the outside mirror in

    EyeGaze
    instance0: right_front_window#0.5
    Instance1: right_outside_mirror#0.5
hypotheses:
    Hypothesis
    confidence: 0.87543
    communicativeFunction:
        Instruct
        requestInfo: car_control
        semanticContent:
            OutsideMirror
            location: right

            ControlState
            state: less
```

Figure 9.6 – The input act after modality fusion

The input message from the speech recogniser is shown in Figure 9.5b. Here, the function of the control command is given by the utterance and represented in the communicative function with the additional information about the desired control state. Furthermore, we know that the driver references an outside mirror. This is also valuable information for modality fusion and is represented in a deictic reference that contains a pattern for an entity of the type *OutsideMirror*.

Since both input messages are received in a very narrow time interval, the modality fusion component (see Section 7.9) of SiAM-dp tries to combine them and finds the missing referenced content of the speech input act in the representation of the eye-gaze input. For this, it examines both hypotheses of the latter for semantic content entities that match the given pattern in the deictic reference. Only one entity, the right outside mirror, is unifiable with the object's type in the pattern. Thus, the mirror is used to resolve the ambiguous content of the speech input act (see Figure 9.6).

9.1.2 Discourse aware Interaction with the Outside Environment

In the second scenario we take the step from in-car interaction to interaction with the outer world, which requires a slightly different approach regarding the model of the environment and the accessed external services. In this approach, the EyeVIUS component provides hypotheses about the buildings that are actually in the driver's focus of attention. If the building is a restaurant, it is possible to retrieve further information and even reserve a table for this restaurant by a natural speech-based dialogue with the system.

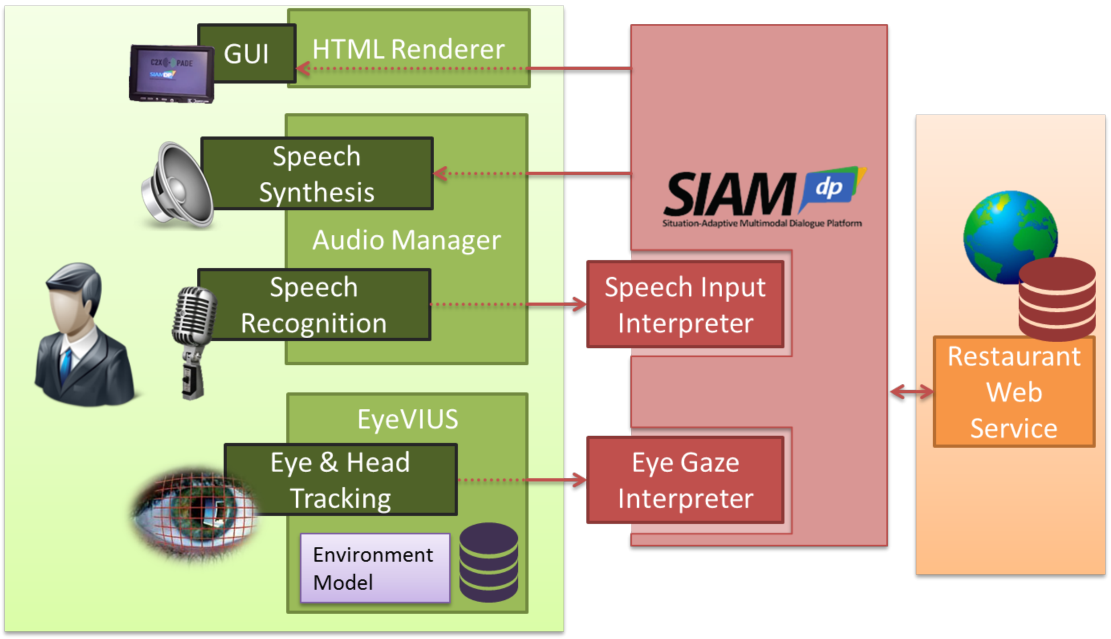


Figure 9.7 – Architecture overview of the restaurant reservation scenario

The necessary information is gained from a restaurant information and reservation web service and redundantly presented to the driver by speech synthesis and GUI output.

Setup

The approach for resolving references to the outside environment used in this scenario is an extension of Moniri et al. (2012). For this, the outside environment was scanned with a professional 3D laser scanner which resulted in a point cloud that was the basis for a 2.5D polygon model. Together with a GPS map-matching algorithm, this model is used to position the vehicle in the environment in real-time. The information from the EyeVIUS system about the driver's attention is then combined with the 2.5D model and the vehicle's position to identify which object in the environment is in the focus of the driver. The result of this reference resolution process serves as input for the dialogue application in the scenario.

Figure 9.7 shows the external components involved in the restaurant scenario. We use the speech synthesis and speech recognition ability of the audio manager for the speech interaction with the driver. Additionally, the EyeVIUS component sends input information about the driver's focus to the dialogue application. The input is processed by the appropriate interpreter components in order to lift syntactical information on a semantic interpretation level before it is handled in the dialogue manager. Table 9.3 shows a classification of the devices integrated in this scenario.

Device	Service	Type	Modality	Device Position	Interaction Range	User number
Eye Tracker and Head tracker	EyeVIUS	Controller	visual	static	far-field	single
Speaker	speech synthesis	Renderer	auditory	static	near-field	multi
Microphone	speech recognition	Controller	auditory	static	near-field	single
In-car display	GUI	Renderer	visual	static	near-field	single

Table 9.3 – Classification of the devices integrated in the restaurant reservation scenario

On the backend side, the application is connected to a web service for restaurant information and reservation via a UMTS internet connection. The content of the received data is mapped on the domain-specific ontology of the application, which contains concepts for the representation of restaurants and information about the actual menu.

Dialogue Phenomena

One main focus in this scenario of the demonstrator is to show the capability of SiAM-dp in handling diverse dialogue phenomena. These are in detail:

Deictic References: Deictic references are part of an utterance that refers to entities that have been introduced into the discourse context in a previous turn or by a different modality. Examples of deictic expressions in the multimodal case have already been shown in the previous scenario. In this scenario the driver starts the restaurant dialogue with the following sequence:

> **Driver:** [gazes at the restaurant] *"What is the name of this restaurant?"*
> **System:** *"The name of the restaurant is AC Café."*

Here, the expression *"this restaurant"* refers to the restaurant entity that is introduced by the driver's gaze via the EyeVIUS component. A second possibility for the referred content of a deictic reference is content from a previous dialogue turn. This is done in the driver's question in the following sequence:

> **Driver:** *"What is the menu for today (for this restaurant)?"*
> **System:** *"Here is the menu for the restaurant AC Café: Fish fillet . . . "*

Figure 9.8 shows how the driver's question is represented in the semantic interpretation. The communicative function is a *SetQuestion* with the attribute *knowledgeItem* set to *menu*. This value specifies the requested information. The reference to the restaurant is described in the first deictic reference which specifies a pattern that restricts the missing entity to an entity of the type *Restaurant*. The second reference is a temporal reference and is described in the next dialogue phenomenon. When passing the fusion and discourse engine (Section 7.9), the deictic reference is resolved with content from

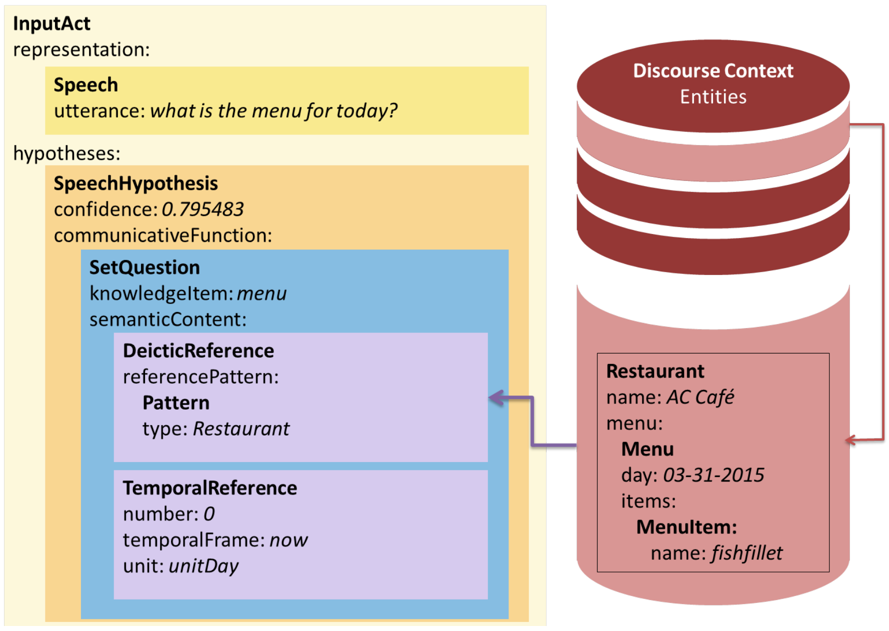


Figure 9.8 – Reference resolution of deictic expressions with entities from the discourse context

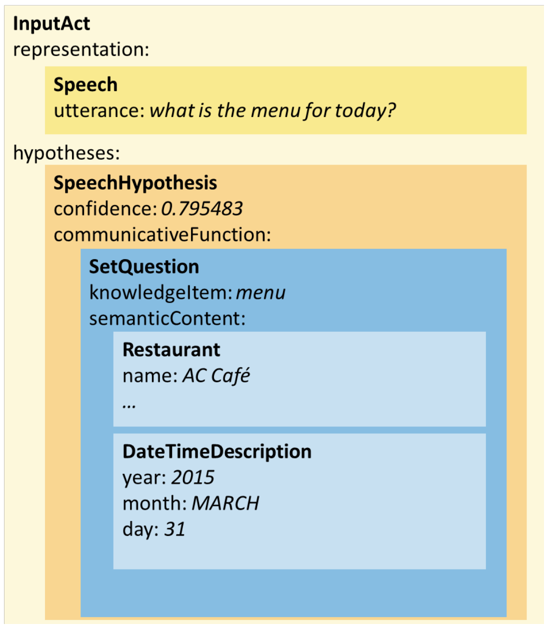
the discourse context. The restaurant introduced in the previous turn lies on top of the internal entity stack of the discourse context and since it matches the specified pattern, it is used to complement the missing content.

Temporal References: Temporal references are used in an utterance in order to specify a point in time mostly relative to a specific temporal fixed-point. The example presented above still contains an unresolved temporal reference. It is the result from the interpretation of the expression *"for today"* and beside the temporal frame *now*, which stands for the actual moment, contains information about the unit and the distance of the referenced point in time to this temporal frame. The FADE component contains algorithms that can automatically resolve temporal references. After the message has passed, this component looks as depicted in Figure 9.9a. Here, both references are resolved allowing the dialogue application to answer the question.

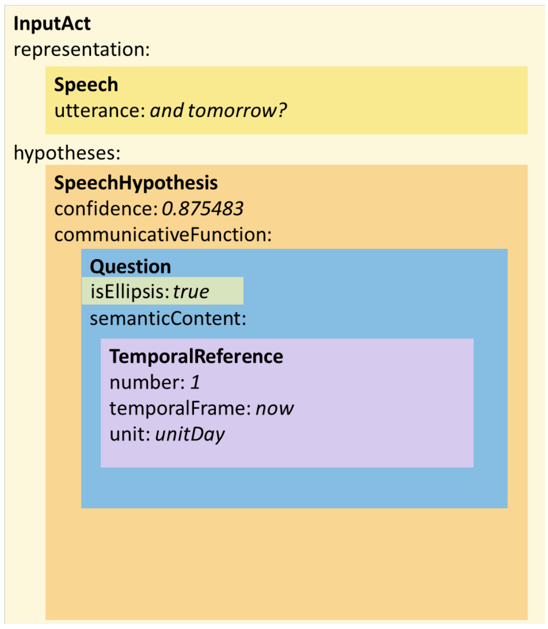
Ellipsis: In an elliptical construction, one or more words of an expression can be omitted, e.g., if some of the constituents have already been mentioned in a previous turn. Nevertheless, the expression makes sense in the current discourse. The following example of the scenario illustrates this:

> **Driver:** *"What is the menu for today?"*
> **System:** *"Here is the menu for the restaurant AC Café: Fish fillet . . . "*
> **Driver:** *"And tomorrow?"*

Here, the second question of the driver *"And tomorrow?"* relates to the first one. The



(a) Result of the temporal reference resolution process



(b) Example for the representation of an elliptic expression

Figure 9.9 – Input act representation examples

stand-alone expression cannot be comprehensively interpreted. Only the implication of the first utterance gives it meaning.

Figure 9.9b shows how an elliptic expression can be represented by the speech input interpreter. Considered separately from the discourse context, the expression *"And tomorrow?"* contains three pieces of information. First, the expression is a question which is represented in the communicative function with the concept *Question*. Secondly the temporal information *"tomorrow"* is a temporal reference that is represented as semantic content. Furthermore, it is very apparent that the expression is an elliptic expression. This is defined with the attribute *isEllisis* that is set to `true`.

The FADE component follows a special approach in processing input that is marked as ellipsis. This approach is depicted in Figure 9.10. Here, the temporal reference is already resolved. The next step is to resolve the ellipsis. The FADE component searches in the previous dialogue acts of the discourse context for a communicative function that is of the same type or a subtype of *Question*. Furthermore, it searches in a matching communicative function for semantic content that is of the same type as the content of the elliptic expression. If a matching instance is found, the original content is replaced with an overlay of the new content with the old content as the background.

In the example, the previously handled *SetQuestion* is a subtype of *Question*. It also contains content of the type *DateTimeDescription*. This content is replaced with the

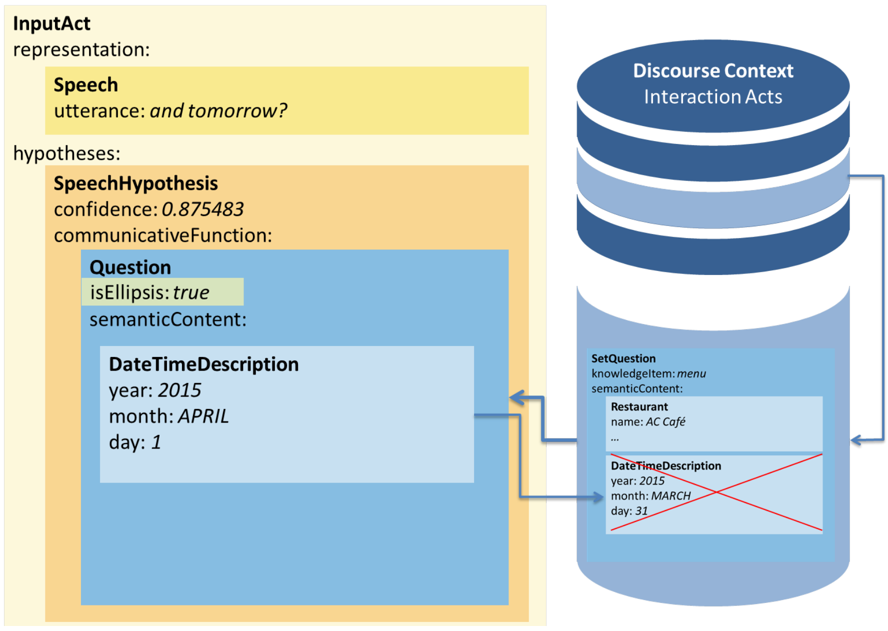


Figure 9.10 – Example for the resolution of elliptic expressions

new content provided by the elliptic expression. The restaurant entity is retained. The result of the resolution process is a complete request that can be answered by the dialogue application.

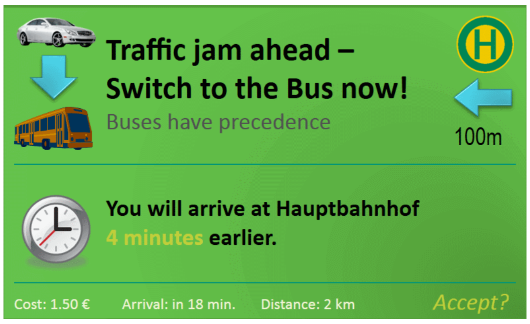
Anaphoric references on system's output: Anaphoric references on entities in the system's output can be resolved in the same way that references to entities in a user's utterance are resolved. For this, it is necessary that the output of the system is also semantically represented. The following sequence is an example:

> **System:** *"Here is the menu for the restaurant AC Café: Fish fillet ..."*
> **Driver:** *"How much is the fish?"*

Besides the utterance of the speech output in syntactic form, the system's response is also represented as a communicative function of the type *Answer*. Furthermore, the presented menu item with the fish is available as an entity in the semantic content. Thus, the entity becomes content of the entity stack in the discourse context and can be referred to by the driver within the next utterance.

9.1.3 Persuasive Travel Assistant

Persuasive technology in the context of cars is employed to help drivers to optimise their driving behaviour. This can be achieved by driving safer, driving more ecologically, saving money on fuel, or reducing travel time. The main goal of a persuasive system is the long-term effect of a change in driver behaviour by exploiting the situational context



Traffic jam ahead – Switch to the Bus now!

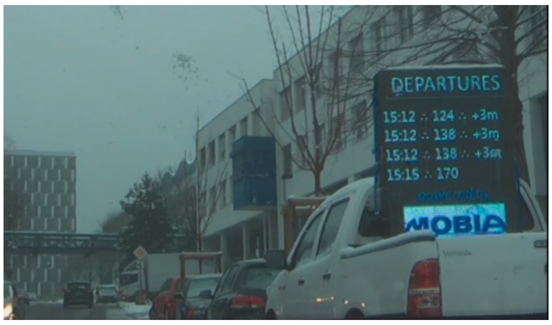
Buses have precedence

100m

You will arrive at Hauptbahnhof 4 minutes earlier.

Cost: 1.50 € Arrival: in 18 min. Distance: 2 km Accept?

(a) In-car screen with the suggestion to switch to the bus and information about the saved travel time



(b) Public billboard with real-time information about the bus schedule

Figure 9.11 – The Persuasive Travel Assistant uses near-field and far-field screens to inform the driver

and giving the right suggestion at the right time. Based on SiAM-dp, a persuasive travel assistant was developed that raises the driver's awareness of the transportation possibilities outside his own car. Learning from drivers' past behaviour in correlation with the driver's agenda, current position, weather and traffic conditions, the travel assistant makes context-aware suggestions for alternative trips, e.g., in case of traffic jams on the route or a lack of parking places around the destination. It finds alternative solutions by combining the trip with public transport or car/bike sharing services available along the selected driving route. Real-time information on public transport or bike sharing services are acquired through external web services. Part of the context, e.g., weather conditions and parking availability, is also provided by external web services.

For this part of the demonstration, two use-cases were selected to show the functionality that is provided by the persuasive component. In the first use-case, a traffic jam situation is resolved by proposing that users take the next bus to their destination. Correlation of bus line and schedule information with underlying route attributes (in this case a bus priority lane on the affected route) and weather conditions results in a context-aware decision that the bus is the best travel option for the driver. Therefore, a persuasive system should provide valuable information, if possible, to assist the driver's decision.

This strategy can sometimes conflict with the goal of avoiding distraction while driving. Showing a lot of background information in the car, especially on a small screen, may have adverse effects and possibly even reduce the effectiveness of the advice. Therefore, the SiAM Persuasive Trip Assistant exploits the strengths of the underlying multimodal dialogue platform and combines the following ideas:

1. The in-car screen space provides the most valuable information.
2. Information is presented both visually and through speech output, thereby allowing the user to choose the most suitable modality depending on the situation.

3. In addition, an external roadside display (electronic billboard) for further complementary information is included.

In this scenario, additional information about the parking availability around the selected bus stop and the ticket price is displayed on the main in-car screen. As a result, the driver receives a suggestion to take a specific bus from the closest bus stop, gets information about the arrival time and ticket price, information about the saved travel time, and parking options. On the outside billboard, real-time bus schedule information was shown so that the driver could further inform himself about alternative options. In the second use-case, a traffic jam situation is resolved by proposing to use a bike-sharing service. In this case, the availability of a bike sharing service, weather conditions and terrain information result in a suggestion to the driver to continue the trip with a bicycle. Information on the saved money on fuel and the arrival time are used as persuasion methods.

Device	Service	Type	Modality	Device Position	Interaction Range	User number
Speaker	speech synthesis	Renderer	auditory	static	near-field	multi
In-car screen	GUI	Renderer	visual	static	near-field	single
Urban display	GUI	Renderer	visual	static	far-field	multi

Table 9.4 – Classification of the devices integrated in the persuasive travel assistant

9.1.4 Distributed Input and Output

The last scenario is a massively multimodal scenario that shows a driver who books cinema tickets directly from his car while he is waiting at a traffic light. During the interaction, several input and output devices are involved that are connected to the dialogue system using a variety of technologies and protocols.

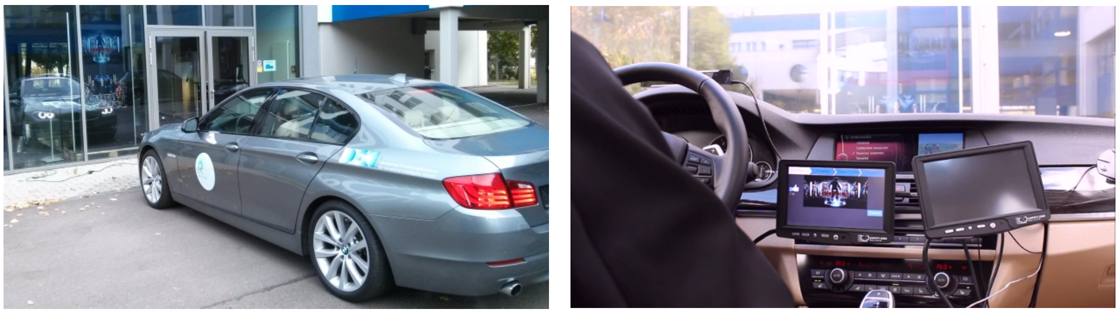


Figure 9.12 – The demonstrator setup. The urban display is simulated by a billboard inside the DFKI building. Two touch screens are mounted at the cockpit of the car.

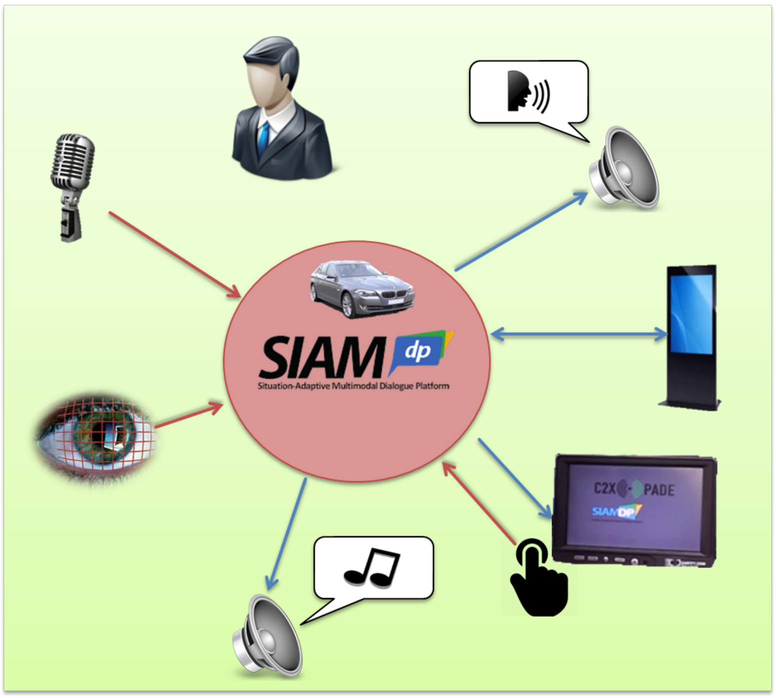


Figure 9.13 – Seven modalities are involved in the communication between driver and dialogue application

The protagonist in this scenario is a driver who is looking at an urban display while he is waiting at a red traffic light. The display and the car are connected via Car2X communication technology, a highly dynamic mobile network where cars and other devices are used as mobile nodes that can create ad-hoc networks (Castronovo, 2013). By using this information source, the dialogue application in the car is aware of the environment and implicitly of the content currently presented on the urban display; in this scenario the posters of two movies. An eye tracker that is installed inside of the car in combination with the EyeVIUS system records eye movements of the driver and recognises if the driver's focus of attention lies on the urban display. In a speech driven dialogue, the driver asks several questions about the movie he is currently looking at, requests a list of cinemas showing this movie, and finally books some tickets for the movie. The answer is distributed via several output modalities. While a video trailer of the movie is shown on the urban display, the associated audio track is played inside the car. Additionally, the dialogue application provides information about the movie via speech output and a display inside the car (Figure 9.12).

In total, seven modalities are involved in the dialogue application (see Figure 9.13 and Table 9.5). For the dialogue system, here arises the challenge to integrate all modalities into the application in a way that they can contribute to one dialogue in a complementary, synchronised and coherent way. The following modalities and technologies are part of the demonstration:

- **Speech input** is realised with the Audio Manager (TCP).
- **Eye-gaze information** is provided by EyeVIUS (UDP).
- **Graphical User Interface (GUI) output** is generated via HTML 5 and AJAX provided by a RESTful web service.
- **Touch input** is realised via touch screen and an HTML 5 user interface (RESTful).
- **Speech output** is generated by the speech synthesis in the Audio Manager (TCP).
- **Audio output** is directly played from the connected Audio Manager. (TCP).
- **Urban display:** The urban display is simulated by a billboard that is connected to a desktop PC. The PC establishes a connection to the car via Car2X.

Device	Service	Type	Modality	Device Position	Interaction Range	User number
Eye Tracker and Head tracker	EyeVIUS	Controller	visual	static	far-field	single
Speaker	speech synthesis	Renderer	auditory	static	near-field	multi
Microphone	speech recognition	Controller	auditory	static	near-field	single
In-car touch display	GUI	Renderer / Controller	visual / tactile	static	near-field	single
Speaker	video soundtrack	Renderer	auditory	static	near-field	single
Urban display	video	Renderer	visual	static	far-field	multi

Table 9.5 – Classification of the devices integrated in the public display scenario

9.2 MADMACS - CeBIT Demonstrator 2015

The MADMACS project (Multi adaptive dialogue management for cyber-physical environments) aims to make a large number of sensors and actuators in the cyber-physical environment available for mobile users, who can interact with them intuitively and at any time in a multimodal fashion.

Just as users change in a Cyber-physical Environment (CPE), the environment should adapt in multiple ways to the user, the task, and the interaction method. Multimodal input and output should allow a free choice of modalities, including near-field and far-field interaction. In all situations, the system should be aware of the user's focus of attention, and be able to guide it using output devices on the user and in the environment.

A first demonstrator of this project was developed for the CeBIT 2015. The system allows one to control actuators in the environment, specifically three lamps and a ventilator. The modality for interaction could be chosen arbitrarily. Thus, the system supports head movements, speech, touchscreens and proximity sensors in a various number of

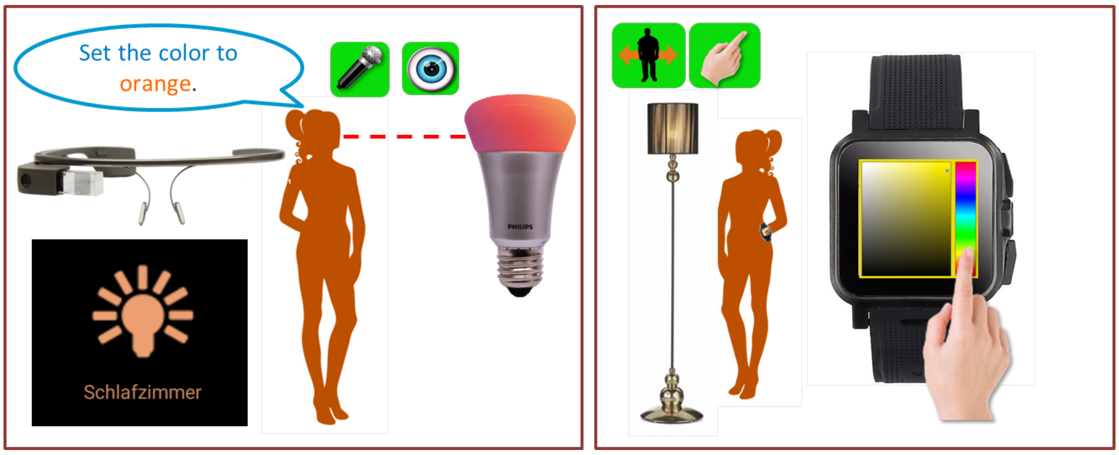


Figure 9.14 – Two multimodal interaction examples of the MADMACS demonstrator. In the first example the lamp colour is changed by a combination of speech input and gaze detection. A graphical feedback is given on the HUD of the Google Glass. The second example embraces the proximity sensor of the smart watch and touch input for the precise selection of the desired colour.

combinations. The device integration and control and the interaction logic were realised by SiAM-dp. Two wearables, Google Glass and a smart watch, are used as interaction devices and are connected to a profoundly multimodal interaction scenario (see Figure 9.14). The supported channels are summarised in Table 9.6.

Two types of actuators are integrated into the demonstrator. The first are LED lights that can be turned on or off and are able to change the colour of their light. The lamps are installed in three different rooms: In the kitchen, the bedroom and the living room. The second actuator type is a fan whose level of speed can be changed remotely.

One of the main aims of the demonstrator is that users can arbitrarily choose the modalities for interaction. Thus, various ways are possible in order to specify the actuator that is actually in focus. One way is to address an actuator directly using speech commands. For the fan, the identification is easy since only one fan is located in the environment. For the identification of the lamp, the speech command must contain additional information about the location of the lamp or the actual colour state, for example *"turn on the lamp in the living room"* or *"change the colour of the red lamp to green"*. Also, more than one lamp can be addressed by speech commands, e.g., by the expressions *"set all lamps to red"* or *"turn off the red lamps"*. Alternatively, the focus of attention can be recognised by the sensor for head rotation of the Google Glass. The feedback of the system is given on the HUD of the glasses in the form of a symbol of the device actual in sight. Another variation is to use a bluetooth proximity sensor that is integrated in the smart watch and to set an actuator into the focus of discourse if the watch is drawn near to it.

Additionally, the control commands can be provided by the use of other modalities. Again, the function can be completely specified by speech input as already demonstrated in the speech utterance examples above.

For more fine-grained statements, other devices are more suitable. The speed level of the fan or the brightness of the lamps can alternatively be set by swipe gestures on the touch pad of the Google Glass. The smart watch allows one to precisely set a concrete colour for a lamp by providing a colour scheme for selection. Such a detailed adjustment is not possible with speech utterances.

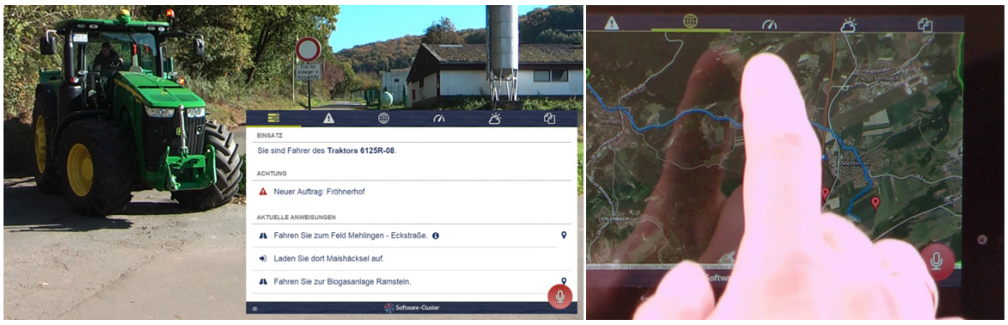
Device	Service	Type	Modality	Device Position	Interaction Range	User number
Google Glass	Microphone Speaker GUI Head Rotation Touchpad	Controller Renderer Renderer Sensor Controller	auditory auditory visual tangible	wearable	near-field	single
Smart Watch	GUI Touchpad Proximity Sensor	Renderer Controller Sensor	visual tangible	wearable	near-field	single
Phillips HUE Lights	lamp control	Actuator	-	static	-	-
Fan	level control	Actuator	-	static	-	-

Table 9.6 – Classification of the devices integrated in the CeBIT demonstrator

9.3 SINNODIUM Demonstrator

In the SINNODIUM project a demonstrator for the multimodal task assignment and introspection in distributed agricultural harvesting processes (Porta et al., 2014b) has been developed. The cloud-based system presented allows one to orchestrate and coordinate a fleet of agricultural machinery and their drivers during an ongoing harvest in case of an unexpected incident. The harvesting process can be replanned in real-time by a management dashboard. The updated instruction is then sent to each affected driver's mobile device. The interaction with the driver must be quick and safe for the driver in order to keep track of a tight schedule. Another factor that is even more crucial is that the driver suffers from a high cognitive load in phases of high concentration, which only leaves little room for additional attention. For such situations multimodal dialogue interaction, including speech and gestures, is highly beneficial.

The multimodal dialogue application in this project is based on SiAM-dp and integrates several channels of a Google Nexus 7 mobile client (see Table 9.7 and Figure 9.15). This includes speech recognition, speech synthesis, and an HTML5 based GUI. In the scenario, the dialogue strategy can be adapted to the current cognitive load of the driver increasing the safety of the driver and ensuring that a new task has been understood.



EINSATZ

Sie sind Fahrer des **Traktors 6125R-08**.

ACHTUNG

Neuer Auftrag: Fröhnerhof

AKTUELLE ANWEISUNGEN

Fahren Sie zum Feld Mehlingen - Eckstraße.

Laden Sie dort Maishäcksel auf.

Fahren Sie zur Biogasanlage Ramstein.

Figure 9.15 – The multimodal interaction with the driver is realised with an android tablet that is mounted in the cockpit of the tractor.

Device	Service	Type	Modality	Device Position	Interaction Range	User number
Tablet	Microphone Speaker GUI	Controller Renderer Controller / Renderer	auditory auditory tactile visual	static	near-field	single

Table 9.7 – Classification of the devices integrated in the SINNODIUM demonstrator

9.4 KOGNIT demonstrator

In the KOGNIT project, the humanoid robot NAO[2] is used as a companion to dementia patients (Prange et al., 2015). His tasks are to continuously monitor the patients' activities and provide cognitive assistance in daily life situations. An integrated speech dialogue functionality allows patients to communicate with NAO through natural language. Thus, the patient can easily ask NAO to remind him to, e.g., take his pills at a specific point in time by speech. Furthermore, the system has access to a knowledge memory that keeps track of the subject's belongings such as keys, glasses, or medications as well as the subject's contacts including people the subject met during the day. Thus, the patient can ask NAO where he can find a specific object in the flat and can request NAO by speech to write a message to a contact in his address book. Here, the system exploits the fact that the contact information is extended with information about relationships making it possible to understand commands like *"write an e-mail to my daughter"*. The control of further actuators in the flat is planned. In the demonstrator, the control of Phillips HUE LED lamps is already supported.

The patient is equipped with a wearable eye tracker from Pupil Labs[3] that can track eye-gazes. This in combination with an integrated camera and image processing algorithms allows one to identify objects on the table that are currently in his focus. NAO shares

[2]https://www.aldebaran.com/en/humanoid-robot/nao-robot
[3]http://pupil-labs.com/

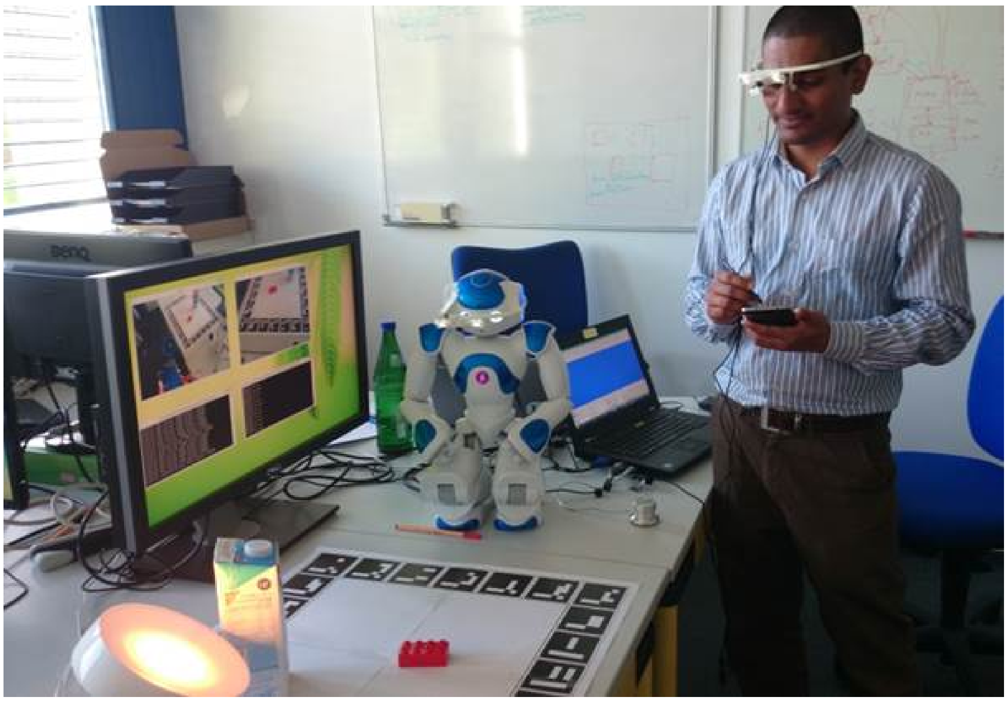


Figure 9.16 – The NAO robot looks at the red block on the table that is currently in focus of the user. The user can start a spoken dialogue and ask questions about the focused element or control the lamp on the table. Another input mode is handwritten text on the smartphone. The messages can be sent to arbitrary people in the user's address book by speech command.

this focus of attention by moving his head and looking at the same object and can answer questions and provide further information about the object (see Figure 9.16).

A further input modality is an interactive pen with which the user writes on the display of a smartphone. The system can automatically distinguish between pen gestures and handwriting and guesses the user's intention. Thus, the user can, alternatively to speech input, write his utterances. Multimodal interaction is also supported. The user can write the message he wants to send to one of his contacts and then give the speech command *"send this to Susan"*. If the recipient is unclear, e.g., if duplicate names exist in the contact list, a clarification dialogue is triggered.

For sending messages, the system actually supports both SMS and e-mail messages, but it is planned to add extensions for social networks. Furthermore, listening services for both messages are steadily running. As soon as a new message is received, the NAO robot gets notified and presents the content to the patient by speech synthesis.

Table 9.8 summarises all devices that are actually included in the demonstrator.

Device	Service	Type	Modality	Device Position	Interaction Range	User number
Nao Robot	body control speech synthesis	Actuator Renderer	- auditory	dynamic	near-field	multi
Pupil – Eye Tracking	eye tracker	Controller	visual	wearable	near-field	single
Smartphone	pen gestures hand-written text speech recognition	Controller Controller Controller	visual visual auditory	dynamic	near-field	single
Phillips HUE light	lamp control	Actuator	-	static	-	-
Message Service	text	Renderer	visual	dynamic	far-field	multi

Table 9.8 – Classification of the devices integrated in the KOGNIT demonstrator

9.5 HySociaTea project

The main research topic of the project HySociaTea (Hybrid Social Teams for Long-Term Collaboration in Cyber-Physical Environments) is the realisation and examination of the collaboration of technological augmented workers with virtual agents, autonomous robots, and soft-bots who work together in a team on common tasks.

In the context of Industrie 4.0, the production of the future becomes more individual with respect to the produced components. In the context of production, the expression “production batch” is defined as the “range of a certain number of units (or assembly units), which are being produced with each set-up of the machine and in the production planning are being followed as one individual unit” (Loudová, 2012). One goal is the batch size 1 production, which means that every produced unit differs from the previous one. For this, a higher flexibility of installation procedures and workflows is required. In order to react quickly on therefore necessary short-term modifications of manufacturing series, component properties, and installation parameters while maintaining efficiency and quality, an appropriate training and optimal support of the workers is essential.

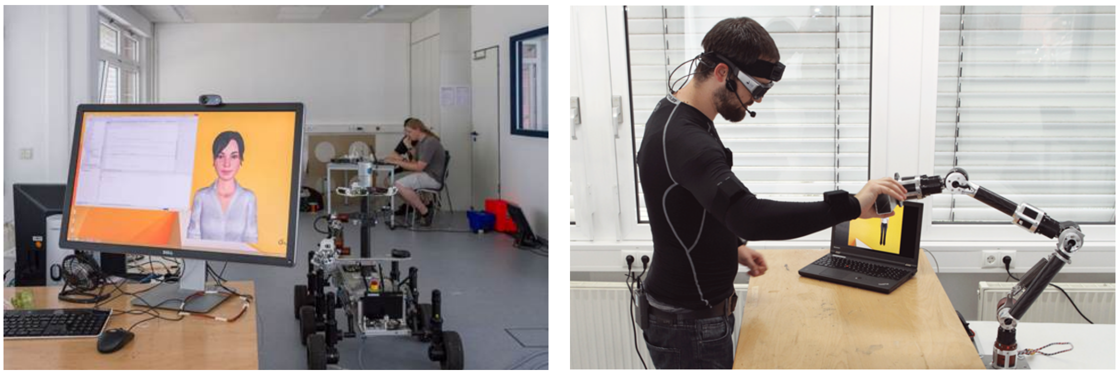


Figure 9.17 – Collaboration of robots, humans, and virtual characters in HySociaTea

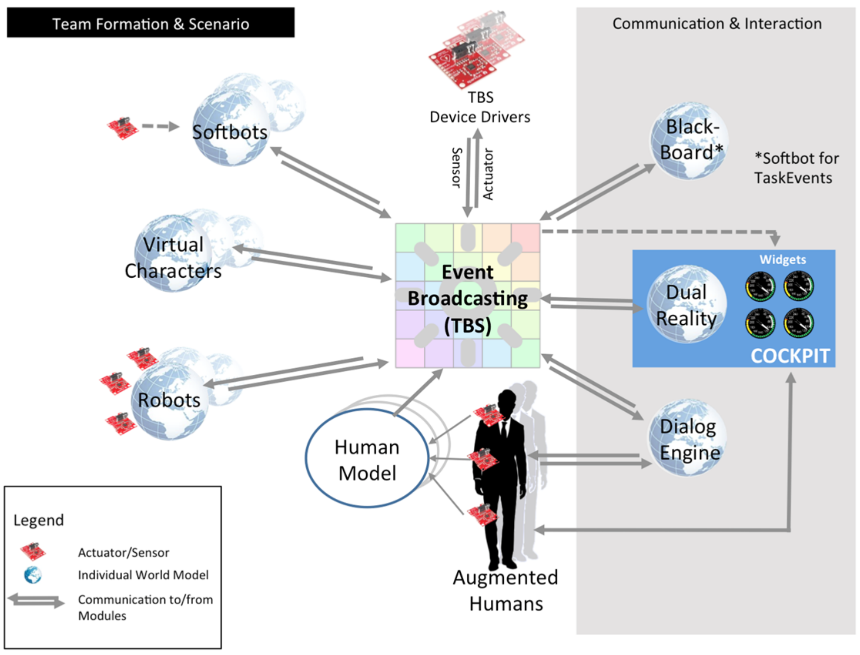


Figure 9.18 – Planned architecture in HySociaTea

The basic idea in HySociaTea is that in the manufacturing environment of the future, human workers will collaborate with robots and virtual characters in a team (see Figure 9.17) that reacts quickly and flexible to new requirements. This especially should allow the team to react to unplanned events by autonomous reorganisation.

The technical systems developed in HySociaTea are mainly meant to be used as assistance-systems for humans working in production-plants. Here, wearable technologies like smart glasses or smart watches may provide valuable information to the workers in a hands-free fashion, e.g., over the HUD display of glasses. Sensors in the environment should help one to understand all physical and communicative interactions of all team members. The team members will be interacting socially and physically. Sharing their individual capabilities for a given sub-task will help the team be able to perform highly challenging or dangerous tasks that would otherwise be infeasible. In order to effectively coordinate a robotic team's actions with those of humans and virtual characters in the team, the robots and virtual characters must be able to effectively convey their planned actions clearly to the human team members and vice versa. Thus, communication is required between all agents as they attempt to solve collaborative multi-agent tasks in a highly dynamic cyber-physical environment.

To create such an intuitive and direct interaction between all involved humans and the artificial entities in the team, integrated sensory information is provided from the human (wearable sensors), the robot (on-board sensors) and the real-world environment equipped with technology (e.g., cameras). All team members in the hybrid team can fulfill various roles: For instance, a robot can serve for the human team member as a physically present companion to naturally interact with, e.g. using speech or gestures. Or the robot can be an active team-member, collaborating to accomplish a physical task (e.g., by helping to lift a certain item or supply a production unit from the store). Other virtual entities, e.g. virtual characters, exist to support straightforward interfacing with the infrastructure. For better integration of the human agents into the whole framework, instrumentation will include direct feedback from sensors or other team members, e.g. using attention recognition and augmented vision. The wearable devices are complemented with other technical devices like tablets or smart phones. The interaction will be multimodal (e.g. using speech and gesture), will have a social nature and will take place between all team members, so that artificial agents interact among one another and with the involved humans.

Figure 9.18 shows the planned architecture in the HySociaTea project. In this architecture, a dialogue engine is responsible for the communication of relevant information between the team members. This module is planned to be realised with SiAM-dp. It will manage the communication between human team members, virtual characters and robots. In this setup, the artificial team members will not only be seen as actuators or sensors of the environment. They take over the role of autonomous and fully-fledged dialogue participants with their own models of the interacting partners, the team as a whole, and the context of the present action. Depending on their current role, they actively contribute to running dialogues, e.g., they answer questions of the human co-workers or accept new commands from them. The technical integration will be realised with the Event Broadcasting System (EBS) (Kahl, 2014), which has been extended with the support of Apache Thrift, an open-source communication protocol for the creation of scalable and interoperable services and is now called Thrift Broadcasting System (TBS). SiAM-dp already contains a component that supports Apache Thrift.

Device	Service	Type	Modality	Device Position	Interaction Range	User number
Wearables	GUI speech recognition speech synthesis	Controller / Renderer	tactile / visual / auditory	wearable	near-field	single
Tablet / Smartphone	GUI speech recognition speech synthesis	Controller Renderer	tactile / visual / auditory	mobile	near-field	single
Screen / Speaker	Virtual Character	Dialogue Participant	auditory / visual	static		
Robots	-	Dialogue Participant	physical	static / mobile	-	-
Sensor Jacket	Body Tracker	Sensor	physical	wearable	near-field	single

Table 9.9 – Classification of the devices integrated in HySociaTea

9.6 Summary

In this chapter applications and demonstrators from various research projects were introduced which have been designed and implemented with the framework and design concepts developed in this thesis. The examples cover a wide spectrum of scenarios and integrated devices and prove the practical relevance of SiAM-dp as well as the flexible field of application.

10 Conclusion and Outlook

10.1 Summary

The goal of this work was the conception, the design and the realisation of a multimodal dialogue platform for the declarative and model-based development of multimodal dialogue applications with a focus on distributed input and output devices in Cyber-physical Environments (CPEs). The main contributions can be divided into three parts:

1. The design of adequate models and strategies for dialogue application specification. This requires the choice of an appropriate meta modelling language which was discussed in Chapter 4. The critical thing for a massively multimodal system is a flexible model for input and output representation which supports arbitrary devices and the description of information on different levels of granularity and meaning. A model that allows this was introduced in Chapter 5. Finally, the models that have been developed for the declarative development of dialogue applications like the specification of projects, dialogue behaviour, speech recognition grammars, and graphical user interfaces were described in detail in Chapter 6.
2. The implementation of a runtime platform that provides a flexible, extendable architecture for the easy integration of new devices and components. The platform deploys several components that implement the concepts and strategies of the previously introduced models and allows one to develop full-fledged multimodal dialogue applications for arbitrary device setups, domains, and scenarios. The runtime platform was presented in Chapter 7.
3. A software development toolkit that is integrated into the Eclipse rich client platform and provides wizards and editors for creating and editing new multimodal dialogue applications based on the above mentioned declarative models. Furthermore, wizards support the creation of new applications and devices from scratch.

Several debugging extensions and monitors support the developers during the development process in searching bugs and testing new features. The available tools were described in Chapter 8.

In Chapter 9, several current and finished research projects were introduced that used the Situation Adaptive Multimodal Dialogue Platform (SiAM-dp) presented in this work for the successful implementation of their scenario specific multimodal demonstrator systems. As a result a large and heterogeneous set of various devices were integrated into the platform.

10.2 Contributions and Results

10.2.1 Research Questions Revisited

In order to highlight the contributions of this work, this section revisits the research questions posed in Section 1.2 and shortly summarises the answers given throughout this thesis.

1. ***Modelling Language:*** *Which requirements must be fulfilled by a meta-modelling language that is used for the declarative development of multimodal dialogue applications?*

   We looked at this issue from several perspectives, examined diverse approaches for the semantic representation of knowledge and language models that are used in related work about multimodal integration and identified several features that should be fulfilled for the model-based development approach. The final choice fell on the Eclipse Modeling Framework (EMF) which was extended with additionally required functionalities and algorithms like cloning, unification, overlay, dynamic data binding, and a model for the definition of patterns.

2. ***Massive Multimodality:*** *How can the massive modality of devices in a CPE be represented in a hierarchical device model and how can this hierarchy be transferred to a structured model for the representation of input and output acts in the communication between the dialogue system and devices?*

   The top level of a type hierarchy for the classification of devices was worked out starting with a partition into sensors, actuators, renderers, and controllers. More concrete device concepts were derived from these top-level concepts. On this basis, the model for the representation of input and output was specified. Outgoing from an abstract class which contains meta-information like timestamps, addressee, and initiator of the message, the device-specific information was structured equivalently to the above mentioned type hierarchy. Thus, devices with an equal type of content were consolidated in one representation which serves as a common interface for them. This is an important factor for device integration allowing one to interchange devices that process the same type of information without the need to adapt the core application.

3. ***Representation of Communicative Meaning:*** *How can interaction between dialogue systems and a highly heterogeneous set of devices in a CPE be represented as modality independent and how can this support the multimodal integration?*

   In related work, several frameworks have been introduced that represent the communicative function or meaning separately from the behaviour respective to realisation. For the multimodal integration, this has two advantages. First, the decisions in dialogue planning can be made purely on the more abstract representation of the dialogue participant's intentions. The actually consulted modalities and surface realisations must not be regarded at this point. This automatically makes the dialogue management independent from the actually used modalities and devices. Thus, the replacement or extension with new devices is realisable without adapting the dialogue management definitions of an application. Second, the description of the communicative meaning introduces a semantic level for the representation of dialogue acts and thus a common representation language for all modalities. In this work we inherited the concepts from the standard for the semantic annotation of dialogue acts (ISO/DIS 24617). Those dialogue acts can also be carriers of semantic content entities deployed within an interaction. We furthermore developed a specific model for the definition of referring expressions that are used to describe unresolved entities with relation to the actual context or cross-modal references that are used for the realisation of modality fusion.

4. ***Declarative Dialogue Application Design:*** *What requirements are demanded of models for the rapid development of systems for multimodal Human-Environment Interaction (HEI) and how can these be satisfied?*

   Several models have been designed for the declarative development of multimodal dialogue applications. The model for dialogue specification is a combination of flowcharts and statecharts and enables application developers to directly integrate the model for input and output description into the workflow. Thus, it is possible to define patterns that indicate on which input messages transitions between dialogue states should be triggered. On the other hand, developers can directly declare the output messages that are emitted. The comprehensive tasks of designing graphical user interfaces and speech recognition grammars is supported by user friendly and easy-to-use models. During the development of both models, the focus was set on hiding the complexity of grammar and Graphical User Interface (GUI) design behind abstract concepts. Thus, they are independent from the grammar specification languages and GUI frameworks actually used for realisation. Furthermore, the models allow one to directly bind named entity rules or graphical components to the corresponding semantic entities they present. The communicative meaning behind complete speech utterances or GUI events can be represented by dialogue act annotations. Hereby, the transformation between syntactic and semantic representation is implicitly defined in the models. For other modalities, the transformations can be declared by mapping rules. The project model consolidates all resources for dialogue applications.

5. ***Dialogue System Architecture:*** *Which type of architecture is required for the realisation of distributed coordinated communication in a CPE?*

   The architecture developed in this thesis was inherited from the dialogue system reference architecture by Maybury and Wahlster (1998a). It was built upon the OSGi framework, an open, modular, and scalable service platform written in Java. The event management was implemented with the *publish-subscribe pattern* and supports a modular architecture approach which is easily adaptable and extendable. Message filtering is based on a unification-based pattern resolution process. It was implemented on the basis of the *OSGi services* functionality. Four layers group the components of the the SiAM-dp platform: Environment layer, core layer, resource and context management layer, and backend layer. The following components were described in detail as part of the core platform: *Dialogue Manager, Fusion & Discourse Resolution, Presentation Planning & Distribution, Speech Recognition Interpreter, GUI Event Interpreter, Project Manager, Session & User Manager, Device Manager, Knowledge Manager, GUI Manager, Speech Grammar Manager.*

6. ***Tool Support:*** *How is an integrated development environment designed that simplifies and accelerates the creation of multimodal dialogue applications?*

   Dialogue application developers are supported with a comprehensive development kit. The development tools were integrated and deployed as plugins for the Eclipse rich client platform. The SiAM-dp development kit comprises the following components:

   - **New Application Wizard** for the creation of new application projects.
   - **New Device Wizard** for the creation of new device components.
   - **General SiAM-dp model editor** for editing domain models.
   - **Grammar rule editor** for editing SiAM-dp specific grammar rules.
   - **Graphical dialogue model editor** for editing workflows.
   - **Application Debug GUI** for monitoring the dialogue workflow and observing the application's context.
   - **HTML 5 Renderer** for prototyping and the dynamic creation of graphical user interfaces.

10.2.2 Related Work Revisited

In this section SiAM-dp is classified regarding the related work presented in Chapter 3.

Multimodal Dialogue Frameworks

Section 3.2 compared several multimodal dialogue frameworks. In Table 10.1, Table 3.1 is extended with an entry for SiAM-dp.

	Discourse Phenomena	Cross-modal References	Fission	Distributed Devices	Physical Acts	Dialogue Specification	Language Standards	Centralised Grammar Management	Extensibility	Reusability	Development Platform
AT&T SMA							x	x		x	
WAMI			x					x		x	
DIANE						x		x	?	x	?
Dialog OS						x				x	x
SmartKom	x	x	x	x	x				x	x	
ODP	x	x	x			x		x	x	x	x
CueMe		?	x	x			x		x	x	x
SiAM-dp	x	x	x	x	x	x	x	x	x	x	x

Table 10.1 – Comparison of SiAM-dp with multimodal dialogue frameworks in the related work

Resolution of Dialogue Discourse Phenomena - Section 7.9 described how dialogue discourse phenomena are resolved in SiAM-dp with the FADE component.

Cross-modal Reference Resolution - The FADE component is also responsible for cross-modal reference resolution (see Section 7.9).

Cross-modal Fission and Presentation Planning - Multimodal fission and presentation planning is realised in the presentation planning component (see Section 7.10)

Distributed Devices - SiAM-dp allows one to flexibly integrate heterogeneous devices of the entire environment (see Section 5.4).

Physical Acts - Physical acts are handled equivalently to communicative acts. Thus, it is possible to syntactically describe physical acts (see Section 5.4) with device-specific representations and semantically annotate them with concepts from the semantic dialogue act model (see Section 5.5).

Dialogue Specification - SiAM-dp supports a statechart and flowchart-based dialogue model for the specification of dialogue behaviour (see Section 6.1).

Language Standards - SiAM-dp incorporates the following standards: SRGS, SCXML, SSML, Agent BML, Semantic Dialogue Annotation Framework.

Centralised Grammar Management - The central Grammar Management Service is responsible for maintaining the supported grammars with respect to the current language and distributing them to the connected speech recogniser (see Section 7.7.1).

Extensibility - The modular architecture of SiAM-dp is built on OSGi and is thus easily extendable with new components and devices (see Section 7.1).

Reusability and Domain Independence - The modelling language of SiAM-dp allows one to easily extend the ontology with domain-specific concepts by Ecore models (see Section 4).

Development Platform - SiAM-dp provides a complete software development kit which is integrated in Eclipse RCP (see Chapter 8).

Multimodal Interaction Modelling

Section 3.2 compared several possibilities for modelling for multimodal interaction. In Table 10.2, Table 3.2 is extended with an entry for SiAM-dp.

Multimodal Representation of Input and Output - Section 5.3 introduced a model taxonomy for the representation of multimodal input and output.

XML-based Language - All Ecore models are persisted or transferred in the form of an XML-based serialisation.

Continuous Refinement of Content - During the data workflow, a message can pass several preprocessing components like interpreters or generators. They continuously refine the message with additional content (see Section 7.2).

Concrete Schemes for Modalities - The modality-specific representations in SiAM-dp are organised in a hierarchical structure which already provides concrete specifications for the most common modalities. New modalities can easily be extended (see 5.4).

Modality Independent Representation of Interaction - The semantic annotation of interaction with dialogue acts as introduced in Section 5.5 is modality independent.

	Input	Output	Multimodal	XML-based	Cont. Refinement	Concrete Schemes	Modality Indepen-dent Representation	Semantic Knowledge Representation	Representation of Uncertainties	Extensibility
M3L	x	x	x	x	x	x	x	x	x	
EMMA	x		x	x	x		x	x	x	x
SWEMMA	x	x	x	x	x		x	x	x	x
SALT	x	x	o	x		x	x	x	x	
SiAM-dp	x	x	x	x	x	x	x	x	x	x

Table 10.2 – Comparison of SiAM-dp with multimodal modelling languages in the related work

Semantic Knowledge Representation - Semantic knowledge is represented with ontologies that are modelled in Ecore (see Chapter 4).

Representation of Uncertainties - Input interpretations are added as hypotheses that contain a confidence attribute (see Section 5.3.2). Quantifiers in the pattern model allow one to express the diverse interpretations of scope ambiguities (see Section 4.4). With the introduction of the reference model (see Section 5.5.1) and the pattern model, which is based on type hierarchies in semantics, it is possible to represent underspecified expressions.

Extensibility - New concepts can easily be derived from the SiAM-dp core concepts.

Dialogue Act Annotation

Section 3.4 compared several languages for the annotation of dialogue acts. In Table 10.3, Table 3.3 is extended with an entry for SiAM-dp.

Representation of Behaviour - In SiAM-dp behaviour is described by the *IORepresentation* taxonomy (see Section 5.4).

Representation of Intention - For the representation of intentions, SiAM-dp uses the model for semantic dialogue acts (see Section 5.5).

Behaviour Interpreter - The interpreter components in SiAM-dp are responsible for lifting syntactic to semantic representation (see Section 5.6).

	Representation of Behaviour	Representation of Intention	Behaviour Interpreter	Behaviour Generator	Situational Adaptive Behaviour	Hierarchy for Communicative Functions
DAMSL		x				
DIT++		x				x
DiAML / ISO 24617-2		x				x
EMMA	x	o				
SAIBA	x	x	x	x	x	
CDE / Virtual Human	x	x	x	x		o
SiAM-dp	x	x	x	x	x	x

Table 10.3 – Dialogue act annotation aspects identified in the related work. The symbol (o) indicates that the aspect has been considered but not deeply elaborated upon.

Behaviour Generator - The generator components generate syntactic from semantic representations (see Section 5.6).

Situational Adaptive Behaviour - The presentation planner chooses a situation dependent on which generator components are applied for the creation of behaviour. Additionally the generators may consider the current context (see Section 7.10).

Hierarchy for Communicative Functions - For the hierarchy of communicative functions, SiAM-dp uses the ISO 24617-2 standard for the annotation of dialogue acts (see Section 5.5).

10.2.3 Scientific Publications

This section summarises the scientific papers that have been published in the context of this work.

Conferences

- **2. Deutscher AAL-Kongress 2009:**
  Robert Neßelrath, C. H. Schulz, J. Schehl, A. Pfalzgraf, N. Pfleger, V. Stein, J. Alexandersson (2009). Homogeneous multimodal access to the digital home for people with cognitive disabilities. In *Ambient Assisted Living 2009. 2. Deutscher AAL-Kongress (AAL-09)*, January 27-28, Berlin, Germany. VDE.
- **11th International Conference on Human-Computer Interaction with Mobile Devices and Services 2009:**
  Daniel Porta, Daniel Sonntag, Robert Neßelrath (2009b). New Business to Business Interaction: Shake your iPhone and speak to it In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services. MobileHCI-09*, September 15 - August 18, Bonn, Germany, ACM
- **3rd International Conference on Health Informatics 2010:**
  Jochen Frey, Christian Husodo Schulz, Robert Neßelrath, Verena Stein, Jan Alexandersson (2010a). Towards pluggable user interfaces for people with cognitive disabilities. In *Proceedings of the 3rd International Conference on Health Informatics. HEALTHINF-2010*, January 20-23, Valencia, Spain. Springer.
- **4. Deutscher AAL-Kongress 2011:**
  Robert Neßelrath, Chensheng Lu, Christian H. Schulz, Jochen Frey, Jan Alexandersson (2011). A gesture based system for context-sensitive interaction with smart homes. In Wichert, R. and Eberhardt, B., editors, *Ambient Assisted Living, 4. AAL-Kongress 2011*, Advanced Technologies and Societal Change, pages 209-219. VDE, Springer.
- **9th International Conference on Intelligent Environments 2013:**
  Robert Neßelrath (2013). Towards a cognitive load aware multimodal dialogue framework for the automotive domain. In *IEEE, editor, Proceedings of the 9th International Conference on Intelligent Environments (IE). International Conference on Intelligent Environments (IE-13)* July 18-19, Athen, Greece. IEEE.
- **10th International Conference on Intelligent Environments 2014:**
  Robert Neßelrath and Michael Feld (2014). Siam-dp: A platform for the model-based development of context-aware multimodal dialogue applications. In *IE'14: Proceedings of the 10th International Conference on Intelligent Environments, Shanghai*, China. IEEE.

- **4th ACM International Symposium on Pervasive Displays:**
Matthieu Deru and Robert Neßelrath (2015). autoUI-ML: A design language for the flexible creation of automotive GUIs based on semantically represented data. In *Gehring, S., Krüger, A., Alt, F., Taylor, N., and Schneegaß, S., editors, Proceedings of the 4th ACM international symposium on pervasive displays*, pages 235-236, Saarbrücken, Germany. ACM.

Workshops

- **6th Workshop on Knowledge and Reasoning in Practical Dialogue Systems 2009:**
Robert Neßelrath, Jan Alexandersson (2009). A 3d gesture recognition system for multimodal dialog systems. In *Proceedings of the 6th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems. Twenty-First International Joint Conference On Artificial Intelligence (IJCAI -09), July 12, Pasadena, California, United States, pages 46-51.* IJCAI 2009.
- **1st International Workshop On Spoken Dialogue Systems Technology 2009:**
Daniel Sonntag, Gerhard Sonnenberg, Robert Neßelrath, Gerd Herzog (2009). Supporting a rapid dialogue engineering process. *Paper presented at the First International Workshop on Spoken Dialogue Systems Technology (IWSDS-2009)*, Kloster Irsee, Germany.
- **4th Workshop on Speech in Mobile and Pervasive Environments 2009:**
Daniel Porta, Daniel Sonntag, Robert Neßelrath (2009a). A multimodal mobile b2b dialogue interface on the iPhone. In *Proceedings of the 4th Workshop on Speech in Mobile and Pervasive Environments in conjunction with MobileHCI '09. SiMPE-09*, September 15, Bonn, Germany.
- **7th Workshop on Knowledge and Reasoning in Practical Dialogue Systems 2009:**
Robert Neßelrath and Daniel Porta (2011). Rapid development of multimodal dialogue applications with semantic models. In *Proceedings of the 7th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (KRPD-11). Twenty-Second International Joint Conference On Artificial Intelligence (IJCAI -11)*, Barcelona, Spain
- **3th Workshop on Cognitive Load and In-Vehicle Human-Machine Interaction:**
Robert Neßelrath and Michael Feld (2013). Towards a cognitive load ready multimodal dialogue system for in-vehicle human-machine interaction. In *Adjunct Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, pages 49-52*, Eindhoven, Netherlands.

Book Contributions

- **A SemProM use case: Health care and compliance:**
  Boris Brandherm, Michael Schmitz, Robert Neßelrath, and Frank Lehmann (2013). In *SemProM: Foundations of Semantic Product Memories for the Internet of Things*, pages 349-361.
- **SiAM-dp, eine multimodale Dialogplattform im Industriekontext:**
  Robert Neßelrath, Tilman Becker, Melanie Reiplinger, and Tim Schwartz (2015). In *Intelligente Vernetzung in der Fabrik - Industrie 4.0 Umsetzungsbeispiele für die Praxis.* Fraunhofer Verlag.

10.3 Future Work

The main goal of this work was the design and realisation of a platform for the development of multimodal dialogue applications in CPEs. The research on this topic covers a very wide spectrum of interdisciplinary research questions. Consequently, this thesis had to address many important research areas, but in the context of this work only some of them could be examined in detail. However, the platform offers the basis and an excellent opportunity for future research and development. The following list presents some possible research questions for future work which are already partly addressed in the follow-up project MadMacs.

- **Strategies for device selection and output planning**

  Presentation planning includes the decision on how output is distributed and coordinated throughout the different available output channels which are dependent on the user's perceptual abilities and preferences. Additionally, in CPEs several activities may take place in parallel, maybe with several users included and in physical proximity to each other. This can potentially imply mutual interference, for example, if output includes audio output, lights or mobile actuators like robots that can block each other. In Section 7.10, the component for presentation planning and distribution is presented which is responsible for the selection of the applied devices and the coordination of output. This base component is a good entry point for further research and the development of rules and strategies that fulfill the task of hardware assignment, presentation planning, and physical actions.

- **Research on different dialogue metaphors**

  The design of an optimal interaction behaviour is dependent on factors like the user, the environment, or the assigned task. The way that the system is represented is important for the way the system is perceived and which interaction metaphors are applied by the user. The CPE can take different roles, e.g., a mentor, an assistant, a business partner, a co-worker or a moderator. The flexible and adaptive nature of the dialogue platform supports the research for adequate interaction metaphors, therefore the behaviour of the system can rapidly be changed for prototyping and the evaluation of usability and user acceptance.

- **Dynamic dialogue model update during runtime**

  In the current version of SiAM-dp, the dialogue model is loaded during the startup process and cannot be manipulated during runtime. For debugging and a dynamic extension of the dialogue workflow, an implementation that allows the replacement or extension of the dialogue model during runtime is a valuable improvement.

- **Extension with sophisticated dialogue management**

  The dialogue manager of the core platform that was introduced in Section 7.3 implements a finite-state based solution based on statecharts. On top of this, more sophisticated dialogue management approaches like frame or information-state based ones (compare Section 2.3.1) can be developed that generate more natural dialogues and are especially useful for collaborative task solving in multi-party scenarios and the realisation of mixed-initiative dialogues. Especially conversations that involve a high number of independent participants (either real participants or virtual ones in the form of, e.g., virtual characters or the system in a specific role) require the management of numerous individual situations and participants' goals that affect the dialogue and therein occurring communicative actions. A purely state-based approach is not an appropriate solution since it is nearly impossible to design every feasible state and transition in such a complex scenario. Furthermore, the design, extension, and maintenance would be extremely laborious and confusing. In order to support advanced, flexible, and mixed initiative interaction with efficient dialogue strategies, SiAM-dp could be extended with information-state and plan-based dialogue managers like *FLoReS (Forward Looking, Reward Seeking)* of Morbini et al. (2014) or the *Conversational Behaviour Generation Framework* of Löckelt (2008).

- **Multi-user interaction in small groups**

  The interaction in CPEs is typically not restricted to a single person. Often more than one user interacts with more than one system and research has to observe n:m-relations where n users can interact with m systems. These constellations can even dynamically change during the runtime of a dialogue application. The sessions manager (see 7.6) already contains models that group users, and the devices they use to participate, into sessions. An interesting question is to retrieve the information for filling this model with content. The research area of attention tracking deals with this topic. For speech input, this includes the questions of who is speaking and with whom. Gestures and multi-touch input must be assigned to the contributing persons. Furthermore, body tracking can help to find out which person in the room interacts with which other dialogue participant (or which Cyber-physical System (CPS)). Therefore the position, but also the line of sight of the user, is relevant. Here, eye-tracking may deliver additional knowledge.

The collected information provides valuable information for dialogue management and presentation planning. It helps to assign incoming input from devices in the environment to the contributing dialogue participants and, with the information of the session model, allows one to identify the intended recipients of information. On the output side, the information can be used for planning the representation of an application. If a communicative act should be provided to a group (the members of a session), the presentation planner can use the knowledge of the session management for selecting all those devices that are necessary in order to reach all participating individuals of the group.

- **Uniform ontology for device description**

  In Chapter 5, a basis approach for the semantic representation of devices in CPEs has been presented. This issue is also addressed in related research areas. Bonino et al. (2008) introduce an ontology for modelling intelligent domotic environments. Beside concepts for the description of rooms and architectural elements, the ontology also supports "controllable" devices with a detailed description of their functionalities and states. This topic also has relevance for middle-ware platforms (Epelde et al., 2011) and smart service architectures for intelligent environments (Frey, 2015). An intended goal would be to develop a homogeneous and commonly accepted standard ontology that raises the interoperability between such systems to a high degree.

Bibliography

Acatech, editor (2011). *Cyber-Physical Systems: Driving Force for Innovation in Mobility, Health, Energy and Production.* Acatech Position. Acatech – National Academy of Science and Engineering, Munich, Germany.

Acatech (2014). Smart Service Welt: Umsetzungsempfehlungen für das Zukunftsprojekt Internetbasierte Dienste für die Wirtschaft.

Aigner, R., Wigdor, D., Benko, H., Haller, M., Lindbauer, D., Ion, A., Zhao, S., and Koh, J. (2012). Understanding mid-air hand gestures: A study of human preferences in usage of gesture types for HCI. *Microsoft Research TechReport MSR-TR-2012-111.*

Alexandersson, J. and Becker, T. (2001). Overlay as the basic operation for discourse processing in a multimodal dialogue system. In *Proceedings of the IJCAI-01 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 1–7.

Alexandersson, J. and Becker, T. (2003). The formal foundations underlying overlay. In *Proceedings of the Fifth International Workshop on Computational Semantics (IWCS-5)*, pages 1–14, Tilburg, The Netherlands.

Alexandersson, J. and Becker, T. (2007). *Efficient Computation of Overlay for Multiple Inheritance Hierachies in Discourse Modeling*, volume 3 of *Studies in Linguistics and Philosophy*, pages 423–455. Kluwer.

Alexandersson, J., Becker, T., and Pfleger, N. (2004). Scoring for overlay based on informational distance. In *Proceedings der 7. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS'04)*, pages 1–4, Vienna, Austria.

Alexandersson, J., Becker, T., and Pfleger, N. (2006). Overlay: The basic operation for discourse processing. In Wahlster, W., editor, *SmartKom: Foundations of Multimodal Dialogue Systems*, Cognitive Technologies, pages 255–267. Springer Berlin Heidelberg.

Allemang, D. and Hendler, J. (2008). *Semantic Web for the Working Ontologist Effective Modeling in RDFS and OWL*. Morgan Kaufmann.

Allen, J. and Core, M. (1997). Draft of DAMSL: Dialog act markup in several layers. Unpublished manuscript.

Allen, J., Ferguson, G., and Stent, A. (2001). An architecture for more realistic conversational systems. In *Proceedings of Intelligent User Interfaces 2001 (IUI-01).*, pages 1–8. Santa Fe, NM.

Alshawi, H. (1990). Resolving quasi logical forms. *Comput. Linguist.*, 16(3):133–144.

Andrew, P., Conard, J., and Woodgate, S. (2005). *Presenting Windows Workflow Foundation*. Sams, Indianapolis, IN, USA.

Atrey, P. K., Hossain, M., El Saddik, A., and Kankanhalli, M. S. (2010). Multimodal fusion for multimedia analysis: A survey. *Multimedia Systems*, 16(6):345–379.

Baddeley, A. (2012). Working memory: Theories, models, and controversies. *Annual review of psychology*, 63:1–29.

Barnett, J., Akolkar, R., Auburn, R., Bodell, M., Burnett, D. C., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R., Raman, T., Reifenrath, K., Rosenthal, N., and Roxendal, J. (2014). State Chart XML (SCXML): State Machine Notation for Control Abstraction - W3C Last Call Working Draft 29.

Benoit, C., Martin, J.-C., Pelachaud, C., Schomkaer, L., and Suhm, B. (2000). Audiovisual and multimodal speech-based systems. In Gibbon, D., Mertins, I., and Moore, R., editors, *Handbook of Multimodal and Spoken Dialogue Systems*, volume 565 of *The Springer International Series in Engineering and Computer Science*, pages 102–203. Springer US.

Bergweiler, S., Deru, M., and Porta, D. (2010). Integrating a multitouch kiosk system with mobile devices and multimodal interaction. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, pages 245–246, New York, NY, USA. ACM.

Bernsen, N. O. (1997). Defining a taxonomy of output modalities from an HCI perspective. *Computer Standards & Interfaces*, 18(67):537 – 553.

Bierwas, I., Dengler, D., Porta, D., Neßelrath, R., and Germesin, S. (2014). Intelligent automated online transaction system for automated interaction with online transaction web sites. EP Patent App. EP20130000558.

Block, H. U., Caspari, R., and Schachtl, S. (2004). Callable manuals - access to product documentation via voice (Anrufbare Bedienungsanleitungen - Zugang zu Produktdokumentation über Sprache). *it - Information Technology*, 46(6):299–305.

Bobbert, D. and Wolska, M. (2007). Dialog OS: An extensible platform for teaching spoken dialogue systems. In *Proceedings of the 11th Workshop on the Semantics and Pragmatics of Dialogue. Trento. Ron Artstein and Laure Vieu*, pages 159–160.

Bolt, R. A. (1980). "Put-that-there": Voice and gesture at the graphics interface. In *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '80, pages 262–270, New York, NY, USA. ACM.

Bonino, D., Castellina, E., and Corno, F. (2008). DOG: An Ontology-Powered OSGi Domotic Gateway. *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, 1:157–160.

Brandherm, B., Schmitz, M., Neßelrath, R., and Lehmann, F. (2013). A SemProM use case: Health care and compliance. In *SemProM: Foundations of Semantic Product Memories for the Internet of Things*, pages 349–361. Springer.

Bui, T. H. (2006). Multimodal dialogue management - state of the art. Technical Report TR-CTIT-06-01, Centre for Telematics and Information Technology, University of Twente, Enschede.

Buitelaar, P., Declerck, T., Frank, A., Racioppa, S., Kiesel, M., Sintek, M., Engel, R., Romanelli, M., Sonntag, D., Loos, B., Micelli, V., Porzel, R., and Cimiano, P. (2006). LingInfo: Design and Applications of a Model for the Integration of Linguistic Information in Ontologies. In *Proceedings of the 5th international conference on Language Resources and Evaluation (LREC)*.

Bunt, H. (2000). Dialogue pragmatics and context specification. In *In Abduction, Belief and Context in Dialogue; studies in computational*, pages 81–150. John Benjamins.

Bunt, H. (2009). The DIT++ taxonomy for functional dialogue markup. In Heylen, D., Pelachaud, C., Catizone, R., and Traum, D., editors, *AAMAS 2009 Workshop, Towards a Standard Markup Language for Embodied Dialogue Acts*, pages 13–24.

Bunt, H. (2011a). Multifunctionality in dialogue. *Journal Computer Speech and Language*, 25(2):222–245.

Bunt, H. (2011b). The semantics of dialogue acts. In *Proceedings of the Ninth International Conference on Computational Semantics*, IWCS '11, pages 1–13, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bunt, H., Alexandersson, J., Carletta, J., Choe, J.-W., Fang, A. C., Hasida, K., Lee, K., Petukhova, V., Popescu-Belis, A., Romary, L., Soria, C., and Traum, D. R. (2010). Towards an ISO Standard for Dialogue Act Annotation. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *LREC*. European Language Resources Association.

Bunt, H., Kipp, M., Maybury, M. T., and Wahlster, W. (2005). Fusion and coordination for multimodal interactive information presentation. In *Multimodal Intelligent Information Presentation*, pages 325–340. Springer.

Carpenter, B. (1992). *The Logic of Typed Feature Structures.* Cambridge University Press, Cambridge.

Cassell, J. (2000). More than just another pretty face: Embodied conversational interface agents. *Communications of the ACM*, 43(4):70–78.

Castronovo, S. (2013). *The Pull Paradigm : foundations of user-centric advanced driver assistant systems based on bidirectional car2X communication.* PhD thesis, Universität des Saarlandes.

Clayberg, E. and Rubel, D. (2008). *Eclipse Plug-ins.* Eclipse Series. Pearson Education.

Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., and Clow, J. (1997). Quickset: Multimodal interaction for distributed applications. In *Proceedings of the Fifth ACM International Conference on Multimedia*, MULTIMEDIA '97, pages 31–40, New York, NY, USA. ACM.

Cohen, P. R., Kaiser, E. C., Buchanan, M. C., Lind, S., Corrigan, M. J., and Wesson, R. M. (2015). Sketch-thru-plan: A multimodal interface for command and control. *Commun. ACM*, 58(4):56–65.

Cohen, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212.

Costa, D. and Duarte, C. (2013). Improving Interaction with TV-Based applications through adaptive multimodal fission. *Emerging Research and Trends in Interactivity and the Human-Computer Interface*, page 54.

Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. M. (1995). Four easy pieces for assessing the usability of multimodal interaction: The CARE properties. In *InterAct*, volume 95, pages 115–120.

Dahl, D. A. (2013). The W3C multimodal architecture and interfaces standard. *Journal on Multimodal User Interfaces*, 7(3):171–182.

Denerz, E. (2013). A formal approach for modelling and implementing agent-based and privacy-aware dialogue management. Master's thesis, Universität des Saarlandes.

Deru, M. and Bergweiler, S. (2014). Swoozy-an innovative design of a distributed and gesture-based semantic television system. In *UBICOMM 2014, The Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 131–139.

Deru, M. and Neßelrath, R. (2015). autoUI-ML: A design language for the flexible creation of automotive GUIs based on semantically represented data. In Gehring, S., Krüger, A., Alt, F., Taylor, N., and Schneegaß, S., editors, *Proceedings of the 4th ACM international symposium on pervasive displays*, pages 235–236, Saarbrücken, Germany. ACM.

Di Fabbrizio, G., Wilpon, J., and Okken, T. (2009). A speech mashup framework for multimodal mobile services. In *Proceedings of the 11th International Conference on Multimodal Interfaces and the 6th Workshop on Machine Learning for Multimodal Interfaces (ICMI-MLMI '09), Cambridge, MA, USA*, pages 71–78.

Dumas, B., Lalanne, D., and Oviatt, S. (2009). Multimodal interfaces: A survey of principles, models and frameworks. In *Human Machine Interaction: Research Results of the MMI Program*, pages 3–26. Springer.

Endres, C. (2012a). *PresTK: Situation-Aware Presentation of Messages and Infotainment Content for Drivers*. PhD thesis, Universität des Saarlandes.

Endres, C. (2012b). Real-time assessment of driver cognitive load as a prerequisite for the situation-aware presentation toolkit PresTK. In *Adjunct Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2012)*, pages 76–79, Portsmouth, New Hampshire, USA.

Epelde, G., Carrasco, E., Zimmermann, G., Alexandersson, J., Neßelrath, R., and Dubielzig, M. (2011). Universal remote console-based next-generation accessible television. *Universal Access in the Information Society*, 10:1–15.

Feld, M. and Müller, C. A. (2011). The automotive ontology: Managing knowledge inside the vehicle and sharing it between cars. In Tscheligi, M., Kranz, M., Weinberg, G., Meschtscherjakov, A., Murer, M., and Wilfinger, D., editors, *AutomotiveUI*, pages 79–86. ACM.

Fensel, D., van Harmelen, F., Horrocks, I., McGuiness, D. L., and Patel-Schneider, P. F. (2001). OIL An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16.

Foster, M. E. (2002). State of the art review: Multimodal fission. *COMIC project Deliverable*, 6(09).

Frey, J. (2015). *ASaP - Integrationsplattform für Smart Services in Intelligenten Umgebungen*. PhD thesis, Universität des Saarlandes.

Frey, J., Schulz, C. H., Neßelrath, R., Stein, V., and Alexandersson, J. (2010a). Towards pluggable user interfaces for people with cognitive disabilities. In *Proceedings of the 3rd International Conference on Health Informatics. HEALTHINF-2010, in Conjunction with Proceedings of the 3rd International Conference on Health Informatics, January 20-23, Valencia, Spain*. Springer.

Frey, J., Stahl, C., Röfer, T., Krieg-Brückner, B., and Alexandersson, J. (2010b). The DFKI competence center for ambient assisted living. In de Ruyter, B., Wichert, R., Keyson, D. V., Markopoulos, P., Streitz, N., Divitini, M., Georgantas, N., and Mana Gomez, A., editors, *Ambient Intelligence: First International Joint Conference, AmI 2010, Málaga, Spain*, volume 6439 of *Lecture Notes in Computer Science*, pages 310–314. Springer, Berlin.

Gallo, L., Placitelli, A. P., and Ciampi, M. (2011). Controller-free exploration of medical image data: Experiencing the kinect. In *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*, pages 1–6. IEEE.

Gebhard, P., Kipp, M., Klesen, M., and Rist, T. (2003). Authoring scenes for adaptive, interactive performances. In *AAMAS*, pages 725–732. ACM.

Gebhard, P., Mehlmann, G., and Kipp, M. (2012). Visual SceneMaker—a tool for authoring interactive virtual characters. *Journal on Multimodal User Interfaces*, 6(1-2):3–11.

Giese, H., Rumpe, B., Schätz, B., and Sztipanovits, J. (2011). Science and engineering of cyber-physical systems (dagstuhl seminar 11441). *Dagstuhl Reports*, 1(11):1–22.

Gillespie-Lynch, K., Greenfield, P. M., Feng, Y., Savage-Rumbaugh, S., and Lyn, H. (2013). A cross-species study of gesture and its role in symbolic development: Implications for the gestural theory of language evolution. *Frontiers in Psychology*, 4(160).

Goldstein, E. B. (2009). *Sensation and perception.* Cengage Learning, eighth edition.

Graetzel, C., Fong, T., Grange, S., and Baur, C. (2004). A non-contact mouse for surgeon-computer interaction. *Technology and Health Care*, 12(3):245–257.

Gruenstein, A., McGraw, I., and Badr, I. (2008). The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proceedings of the 10th international conference on Multimodal interfaces*, ICMI '08, pages 141–148, New York, NY, USA. ACM.

Gurevych, I., Porzel, R., Slinko, E., Pfleger, N., Alexandersson, J., and Merten, S. (2003). Less is more: Using a single knowledge representation in dialogue systems. In *Proceedings of the HLT-NAACL 2003 Workshop on Text Meaning*, pages 14–21.

Gutiérrez, J. and Horrillo, M. (2014). Advances in artificial olfaction: Sensors and applications. *Talanta*, 124:95 – 105.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3).

Herzog, G. and Reithinger, N. (2006). The SmartKom architecture: A framework for multimodal dialogue systems. In Wahlster (2006b), pages 55–70.

Hillairet, G., Bertrand, F., Lafaye, J. Y., et al. (2008). Bridging EMF applications and RDF data sources. In *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering, SWESE2008.*

Hirsch, M., Cheng, J., Reiss, A., Sundholm, M., Lukowicz, P., and Amft, O. (2014). Hands-free gesture control with a capacitive textile neckband. In *Proceedings of the 2014 ACM International Symposium on Wearable Computers*, pages 55–58. ACM.

Hoepfinger, J. and Candell, E. (2010). Voice extensible markup language (VoiceXML) 3.0 requirements. World Wide Web Consortium, Working Draft.

Hofweber, T. (2014). Logic and ontology. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Standford University, fall 2014 edition.

Honold, F., Schüssel, F., and Weber, M. (2012). Adaptive probabilistic fission for multimodal systems. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pages 222–231. ACM.

Hurum, S. (1988). Handling scope ambiguities in english. In *Proceedings of the Second Conference on Applied Natural Language Processing*, ANLC '88, pages 58–65, Stroudsburg, PA, USA. Association for Computational Linguistics.

ISO 24617-2:2012 (2012). *Language resource management – Semantic annotation framework (SemAF) – Part 2: Dialogue acts.* ISO, Geneva, Switzerland.

Jacob, M. G., Wachs, J. P., and Packer, R. A. (2013). Hand-gesture-based sterile interface for the operating room using contextual cues for the navigation of radiological images. *Journal of the American Medical Informatics Association*, 20(e1):e183–e186.

Jaimes, A. and Sebe, N. (2007). Multimodal human-computer interaction: A survey. *Computer Vision and Image Understanding*, 108(1-2):116–134.

Johnston, M. (2009). Building multimodal applications with EMMA. In *Proceedings of the 11th International Conference on Multimodal Interfaces and the 6th Workshop on Machine Learning for Multimodal Interfaces (ICMI-MLMI '09), Cambridge, MA, USA*, pages 47–54.

Johnston, M., Baggia, P., Burnett, D. C., Carter, J., Dahl, D. A., McCobb, G., and Raggett, D. (2009). EMMA: Extensible MultiModal Annotation Markup Language - W3C Recommendation 10 February 2009.

Jokinen, K. and Wilcock, G. (2014). Multimodal open-domain conversations with the NAO robot. In Mariani, J., Rosset, S., Garnier-Rizet, M., and Devillers, L., editors, *Natural Interaction with Robots, Knowbots and Smartphones*, pages 213–224. Springer New York.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing*, chapter 24, pages 863–891. Pearson Education International, 2nd edition.

Just, M. A. and Carpenter, P. A. (1976). The role of eye-fixation research in cognitive psychology. *Behavior Research Methods & Instrumentation*, 8(2):139–143.

Kagermann, H., Wahlster, W., and Helbig, J., editors (2013). *Deutschlands Zukunft als Produktionsstandort sichern: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0, Abschlussbericht des Arbeitskreises Industrie 4.0.* Forschungsunion im Stifterverband für die Deutsche Wirtschaft e.V., Berlin.

Kahl, G. (2014). *Dual Reality Framework: Basistechnologien zum Monitoring und Steuern von cyber-physischen Umgebungen.* PhD thesis, Universität des Saarlandes.

Karam, M. (2009). *A Framework for Gesture-based Human Computer Interactions.* VDM Verlag, Saarbrücken, Germany.

Karray, F., Alemzadeh, M., Saleh, J. A., and Arab, M. N. (2008). Human-computer interaction: Overview on state of the art. *International Journal on Smart Sensing and Intelligent Systems*, 1(1):137–159.

Kelly, S. D., Barr, D. J., Church, R. B., and Lynch, K. (1999). Offering a hand to pragmatic understanding: The role of speech and gesture in comprehension and memory. *Journal of Memory and Language*, 40(4):577 – 592.

Kempe, B., Pfleger, N., and Löckelt, M. (2005). Generating verbal and nonverbal utterances for virtual characters. In Subsol, G., editor, *International Conference on Virtual Storytelling*, volume 3805 of *Lecture Notes in Computer Science*, pages 73–76. Springer.

Kendon, A., Sebeok, T. A., and Umiker-Sebeok, J. (1981). *Nonverbal communication, interaction, and gesture: Selections from Semiotica*, volume 41. Walter de Gruyter.

Kern, D., Mahr, A., Castronovo, S., Schmidt, A., and Müller, C. (2010). Making use of drivers' glances onto the screen for explicit gaze-based interaction. In *Proceedings of the 2Nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '10, pages 110–116, New York, NY, USA. ACM.

Kühnel, C., Westermann, T., Hemmert, F., Kratz, S. G., Müller, A., and Möller, S. (2011). I'm home: Defining and evaluating a gesture set for smart-home control. *Int. J. Hum.-Comput. Stud.*, 69(11):693–704.

Knapp, M. and Hall, J. (2009). *Nonverbal communication in human interaction.* Cengage Learning.

Kobsa, A., Allgayer, J., Reddig, C., Reithinger, N., Schmauks, D., Harbusch, K., and Wahlster, W. (1986). Combining deictic gestures and natural language for referent identification. In *COLING*, pages 356–361.

Koons, D. B., Sparrell, C. J., and Thórisson, K. R. (1991). Integrating simultaneous input from speech, gaze, and hand gestures. In Maybury, M. T., editor, *AAAI Workshop on Intelligent Multimedia Interfaces*, pages 257–276. AAAI Press / The MIT Press.

Kopp, S., Krenn, B., Marsella, S., Marshall, A. N., Pelachaud, C., Pirker, H., Thórisson, K. R., and Vilhjálmsson, H. H. (2006). Towards a common framework for multimodal generation: The behavior markup language. In Gratch, J., Young, M., Aylett, R., Ballin, D., and Olivier, P., editors, *IVA*, volume 4133 of *Lecture Notes in Computer Science*, pages 205–217. Springer.

Krieg-Brückner, B., Röfer, T., Shi, H., and Gersdorf, B. (2010). Mobility assistance in the bremen ambient assisted living lab. *GeroPsych: The Journal of Gerontopsychology and Geriatric Psychiatry*, 23(2):121 – 130.

Krieger, H.-U. and Schäfer, U. (1994). TDL-A Type Description Language for Constraint-Based Grammars. In *COLING*, pages 893–899.

Lalanne, D., Nigay, L., Palanque, P., Robinson, P., Vanderdonckt, J., and Ladr, J.-F. (2009). Fusion engines for input multimodal interfaces: A survey. In *IMCI-MLMI '09: Proceedings of the 11th International Conference on Multimodal Interfaces and the 6th Workshop on Machine Learning for Multimodal Interfaces, Cambridge, MA, USA*, pages 153–160.

Löckelt, M. (2008). *A Flexible and Reusable Framework for Dialogue and Action Management in Multi-Party Discourse.* PhD thesis, Universität des Saarlandes.

Lee, E. A. (2008). Cyber physical systems: Design challenges. In *Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC '08), Orlando, FL, USA*, pages 363–369.

Lison, P. (2012). Probabilistic dialogue models with prior domain knowledge. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, SIGDIAL '12, pages 179–188, Stroudsburg, PA, USA. Association for Computational Linguistics.

Löckelt, M., Pfleger, N., and Reithinger, N. (2007). Multi-party conversation for mixed reality. *International Journal of Virtual Reality*, 6(4):31–42.

Loudová, B. (2012). Influence of production batch size on companys logistic costs. In *Proceedings in EIIC-1st Electronic International Interdisciplinary Conference.*

MacDougall, W. (2013). *Industrie 4.0: Smart Manufacturing for the Future.* Germany Trade and Invest, Gesellschaft für Außenwirtschaft und Standortmarketing mbH, Berlin.

Mahr, A., Endres, C., Schneeberger, T., and Müller, C. (2011). Determining human-centered parameters of ergonomic micro-gesture interaction for drivers using the theatre approach. In *Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications, 3rd, December 1-2, Salzburg, Austria.* ACM.

Marshall, S. P. (2007). Identifying cognitive state from eye metrics. *Aviation, Space, and Environmental Medicine*, 78(5).

Matsukura, H., Yoneda, T., and Ishida, H. (2013). Smelling screen: Development and evaluation of an olfactory display system for presenting a virtual odor source. *Visualization and Computer Graphics, IEEE Transactions on*, 19(4):606–615.

Maybury, M. T. and Wahlster, W. (1998a). An introduction to intelligent user interfaces. In Maybury and Wahlster (1998b), pages 1–13.

Maybury, M. T. and Wahlster, W., editors (1998b). *Readings in Intelligent User Interfaces.* Morgan Kaufmann, San Francisco, CA.

Meurant, L. (2008). The speaker's eye gaze: Creating deictic, anaphoric and pseudo-deictic spaces of reference. *Sign Languages: Spinning and Unraveling the Past, Present and Future. TISLR*, 9:403–414.

Milward, D. and Beveridge, M. (2003). Ontology-based dialogue systems. In *Proceedings of the 3rd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Acapulco, Mexico*, pages 9–18.

Mitrevska, M., Moniri, M. M., Neßelrath, R., Schwartz, T., Feld, M., Körber, Y., Deru, M., and Müller, C. (2015). SiAM - Situtation-adaptive multimodal interaction for innovative mobility concepts of the future. In *IE'15: Proceedings of the 11th International Conference on Intelligent Environments*, Prague, Czech Republic. IEEE.

Moniri, M. M., Feld, M., and Müller, C. (2012). Personalized in-vehicle information systems: Building an application infrastructure for smart cars in smart spaces. In *Intelligent Environments (IE), 2012 8th International Conference on*, pages 379–382.

Moniri, M. M. and Müller, C. (2012). Multimodal reference resolution for mobile spatial interaction in urban environments. In *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '12, pages 241–248, New York, NY, USA. ACM.

Moniri, M. M. and Müller, C. (2014). Eyevius: Intelligent vehicles in intelligent urban spaces. In *Adjunct Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 1–6, New York, NY, USA. ACM.

Morbini, F., Devault, D., Sagae, K., Gerten, J., Nazarian, A., and Traum, D. (2014). FLoReS: A forward looking, reward seeking, dialogue manager. In *Natural Interaction with Robots, Knowbots and Smartphones - Putting Spoken Dialog Systems into Practice*, pages 313–325. Sp.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26. Publisher: John Benjamins Publishing Company.

National Joint Committee for the Communication Needs of Persons with Severe Disabilities (1992). Guidelines for meeting the communication needs of persons with severe disabilities.

Neal, J. G., Thielman, C. Y., Dobes, Z., Haller, S. M., and Shapiro, S. C. (1989). Natural language with integrated deictic and graphic gestures. In *Proceedings of the*

Workshop on Speech and Natural Language, HLT '89, pages 410–423, Stroudsburg, PA, USA. Association for Computational Linguistics.

Neßelrath, R. (2013). Towards a cognitive load aware multimodal dialogue framework for the automotive domain. In *Proceedings of the 9th International Conference on Intelligent Environments (IE). International Conference on Intelligent Environments (IE-13), July 18-19, Athen, Greece.* IEEE.

Neßelrath, R. and Alexandersson, J. (2009). A 3D gesture recognition system for multimodal dialog systems. In *Proceedings of the 6th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems. Twenty-First International Joint Conference On Artificial Intelligence (IJCAI -09), July 12, Pasadena, California, United States*, pages 46–51. IJCAI 2009.

Neßelrath, R. and Feld, M. (2013). Towards a cognitive load ready multimodal dialogue system for in-vehicle human-machine interaction. In *Adjunct Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, pages 49–52, Eindhoven, Netherlands.

Neßelrath, R. and Feld, M. (2014). SiAM-dp: A platform for the model-based development of context-aware multimodal dialogue applications. In *IE'14: Proceedings of the 10th International Conference on Intelligent Environments*, Shanghai, China. IEEE.

Neßelrath, R., Lu, C., Schulz, C. H., Frey, J., and Alexandersson, J. (2011). A gesture based system for context-sensitive interaction with smart homes. In Wichert, R. and Eberhardt, B., editors, *Ambient Assisted Living, 4. AAL-Kongress 2011*, Advanced Technologies and Societal Change, pages 209–219, Berlin, Germany. VDE, Springer.

Neßelrath, R. and Porta, D. (2011). Rapid development of multimodal dialogue applications with semantic models. In *Proceedings of the 7th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (KRPD-11). Twenty-Second International Joint Conference On Artificial Intelligence (IJCAI -11)*, Barcelona, Spain.

Neßelrath, R., Schulz, C. H., Schehl, J., Pfalzgraf, A., Pfleger, N., Stein, V., and Alexandersson, J. (2009). Homogeneous multimodal access to the digital home for people with cognitive disabilities. In *Ambient Assisted Living, 2. AAL-Kongress 2009*, Berlin, Germany. VDE.

Neßelrath, R., Becker, T., Reiplinger, M., and Schwartz, T. (2015). SiAM-dp, eine multimodale Dialogplattform im Industriekontext. In *Intelligente Vernetzung in der Fabrik - Industrie 4.0 Umsetzungsbeispiele für die Praxis*, pages 277–288. Fraunhofer Verlag.

Nigay, L. and Coutaz, J. (1993). A design space for multimodal systems: Concurrent processing and data fusion. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 172–178, New York, NY, USA. ACM.

Obaid, M., Kistler, F., Häring, M., Bühling, R., and André, E. (2014). A framework for user-defined body gestures to control a humanoid robot. *I. J. Social Robotics*, 6(3):383–396.

Oberle, D., Ankolekar, A., Hitzler, P., Cimiano, P., Sintek, M., Kiesel, M., Mougouie, B., Baumann, S., Vembu, S., Romanelli, M., Buitelaar, P., Engel, R., Sonntag, D., Reithinger, N., Loos, B., Zorn, H.-P., Micelli, V., Porzel, R., Schmidt, C., Weiten, M., Burkhardt, F., and Zhou, J. (2007). DOLCE ergo SUMO: On foundational and domain models in the SmartWeb integrated ontology (SWIntO). *Web Semantics*, 5(3):156–174.

Openstream (2011). Enterprise mobility: Benefits of multimodality in mobile force automation. Technical report, Openstream Inc.

Openstream (2015). Cue-me: Context-aware and multimodal mobile development platform. `http://www.openstream.com/cueme/` (retrieved Aug. 31, 2015).

Ovchinnikova, E. (2012). *Integration of World Knowledge for Natural Language Understanding*, chapter Natural Language Understanding and World Knowledge, pages 15–37. Atlantis Press.

Oviatt, S. (1996). Multimodal interfaces for dynamic interactive maps. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '96, pages 95–102, New York, NY, USA. ACM.

Oviatt, S. (1999a). Mutual disambiguation of recognition errors in a multimodel architecture. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 576–583, New York, NY, USA. ACM.

Oviatt, S. (1999b). Ten myths of multimodal interaction. *Communications of the ACM*, 42(11):74–81.

Oviatt, S. (2012). Multimodal interfaces. In *The Human Computer Interaction Handbook*, chapter 18, pages 405–430. Crc Pr Inc.

Oviatt, S., Cohen, P., Wu, L., Vergo, J., Duncan, L., Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J., and Ferro, D. (2000). Designing the user interface for multimodal speech and pen-based gesture applications: State-of-the-art systems and future research directions. *Hum.-Comput. Interact.*, 15(4):263–322.

Oviatt, S. L. and Cohen, P. R. (2015a). Commercialization of multoimodal interfaces. In *The paradigm shift to multimodality in contemporary computer interfaces*, chapter 9. Morgan & Claypool Publishers.

Oviatt, S. L. and Cohen, P. R. (2015b). *The Paradigm Shift to Multimodality in Contemporary Computer Interfaces.* Morgan & Claypool Publishers.

Oviatt, S. L., Lunsford, R., and Coulston, R. (2005). Individual differences in multimodal integration patterns: What are they and why do they exist? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 241–249, New York, NY, USA. ACM.

Perrault, C. R. and Allen, J. (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, 6(3-4):167–182.

Petukhova, V. and Bunt, H. (2012). The coding and annotation of multimodal dialogue acts. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Petukhova, V. V. (2011). *Multidimensional dialogue modelling.* PhD thesis, Tilburg University.

Pfleger, N. (2007). *Context-based Multimodal Interpretation: An Integrated Approach to Multimodal Fusion and Discourse Processing.* PhD thesis, Universität des Saarlandes.

Pfleger, N. and Alexandersson, J. (2006). Towards resolving referring expression by implicitly activated referents in practical dialogue systems. In *The Proceedings of the 10th Workshop on the Semantics and Pragmaticsof Dialogue - BRANDIAL06*, pages 2–9, Potsdam, Germany.

Pfleger, N., Alexandersson, J., and Becker, T. (2002). Scoring functions for overlay and their application in discourse processing. In *Proceedings of KONVENS 2002*, pages 139–146.

Pfleger, N. and Schehl, J. (2006). Development of advanced dialog systems with PATE. In *Proceedings of Interspeech 2006—ICSLP: 9th International Conference on Spoken Language Processing, Pittsburgh, PA, USA*, pages 1778–1781.

Poesio, M. (1994). Ambiguity, underspecification and discourse interpretation. In *Proceedings of the First International Workshop on Computational Semantics*, pages 151–160.

Porta, D., Deru, M., Bergweiler, S., Herzog, G., and Poller, P. (2014a). *Building Multimodal Dialog User Interfaces in the Context of the Internet of Services*, pages 145–162. Cognitive Technologies. Springer.

Porta, D., Sonntag, D., and Neßelrath, R. (2009a). A multimodal mobile B2B dialogue interface on the iPhone. In *Proceedings of the 4th Workshop on Speech in Mobile and Pervasive Environments in conjunction with MobileHCI '09. SiMPE-09, September 15, Bonn, Germany.*

Porta, D., Sonntag, D., and Neßelrath, R. (2009b). New business to business interaction: Shake your iphone and speak to it. In *Proceedings of the 11th International Conference on Human-ComputerInteraction with Mobile Devices and Services. MobileHCI-09, September15 - August 18, Bonn, Germany.* ACM.

Porta, D., Tuncer, Z., Wirth, M., and Hellenschmidt, M. (2014b). Multimodal task assignment and introspection in distributed agricultural harvesting processes. In *Proceedings of the Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2014), Rome, Italy*, pages 227–232. International Academy, Research and Industry Association (IARIA).

Prange, A., Sandrala, I. P., Weber, M., and Sonntag, D. (2015). Robot companions and smartpens for improved social communication of dementia patients. In Brdiczka, O., Chau, P., Carenini, G., Pan, S., and Kristensson, P. O., editors, *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion*, IUI Companion '15, pages 65–68, New York, NY, USA. ACM.

Qvarfordt, P. (2005). Conversing with the user based on eye-gaze patterns. In *In Proceedings of CHI05*, pages 221–230. ACM Press.

Rautaray, S. S. and Agrawal, A. (2015). Vision based hand gesture recognition for human computer interaction: A survey. *Artificial Intelligence Review*, 43(1):1–54.

Reithinger, N., Gebhard, P., Löckelt, M., Ndiaye, A., Pfleger, N., and Klesen, M. (2006). VirtualHuman: Dialogic and affective interaction with virtual characters. In *Proceedings of the 8th International Conference on Multimodal Interfaces*, ICMI '06, pages 51–58, New York, NY, USA. ACM.

Riener, A. (2012). Gestural interaction in vehicular applications. *Computer*, 45(4):42–47.

Ruesch, J. and Kees, W. (1956). *Nonverbal communication: Notes on the visual perception of human relations.* University of California Press.

Schehl, J., Pfalzgraf, A., Pfleger, N., and Steigner, J. (2008). The BabbleTunes system—talk to your iPod! In *Proceedings of the 10th International Conference on Multimodal Interfaces (ICMI '08), Chania, Crete, Greece*, pages 77–80.

Schmitz, M., Baus, J., and Dörr, R. (2008). The digital sommelier: Interacting with intelligent products. In Floerkemeier, C., Langheinrich, M., Fleisch, E., Mattern, F., and Sarma, S., editors, *The Internet of Things*, volume 4952 of *Lecture Notes in Computer Science*, pages 247–262. Springer Berlin Heidelberg.

SemVox, G. (2015). ODP S3 - The leading dialog and assistance technology. `http://www.semvox.de/uploads/pdf/SEM_Produktblatt_FINAL_Web_EN.pdf`. Accessed: 2015-10-14.

Serrano, M. and Nigay, L. (2009). Temporal aspects of care-based multimodal fusion: From a fusion mechanism to composition components and woz components. In *Proceedings of the 2009 International Conference on Multimodal Interfaces*, ICMI-MLMI '09, pages 177–184, New York, NY, USA. ACM.

Shukla, D. and Schmidt, B. (2006). *Essential Windows workflow foundation.* Addison-Wesley Professional.

Sibert, L. E. and Jacob, R. J. K. (2000). Evaluation of eye gaze interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 281–288, New York, NY, USA. ACM.

Siroux, J., Guyomard, M., Multon, F., and Remondeau, C. (1995). Modeling and processing of oral and tactile activities in the georal system. In Bunt, H., Beun, R.-J., and Borghuis, T., editors, *Multimodal Human-Computer Communication*, volume 1374 of *Lecture Notes in Computer Science*, pages 101–110. Springer.

Song, D. (2006). *Combining Speech User Interfaces of Different Applications.* PhD thesis, Ludwig-Maximilians-Universität München.

Sonntag, D., Engel, R., Herzog, G., Pfalzgraf, A., Pfleger, N., Romanelli, M., and Reithinger, N. (2007). SmartWeb handheld—multimodal interaction with ontological knowledge bases and semantic web services. In *Artifical Intelligence for Human Computing: ICMI 2006 and IJCAI 2007 International Workshops*, pages 272–295. Springer.

Sonntag, D. and Möller, M. (2009). Unifying semantic annotation and querying in biomedical images repositories. In *Proceedings of the International Conference on Knowledge Management and Information Sharing (KMIS), Madeira, Portugal.*

Sonntag, D., Neßelrath, R., Sonnenberg, G., and Herzog, G. (2009). Supporting a rapid dialogue engineering process. Paper presented at the First International Workshop on Spoken Dialogue Systems Technology (IWSDS-2009), Kloster Irsee, Germany.

Sonntag, D. and Romanelli, M. (2006). A multimodal result ontology for integrated semantic web dialogue applications. In *Proceedings of the 5th Conference on Language Resources and Evaluation(LREC 2006)*, Genova, Italy.

Starker, I. and Bolt, R. A. (1990). A gaze-responsive self-disclosing display. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 3–10, New York, NY, USA. ACM.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2009). *EMF: Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2nd edition.

Stiefelhagen, R. and Yang, J. (1997). Gaze tracking for multimodal human-computer interaction. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 4, pages 2617–2620 vol.4.

Stock, O., Strapparava, C., and Zancanaro, M. (1996). Human-computer interaction through natural language and hypermedia in alfresco. *SIGCHI Bull.*, 28(3):102–107.

Tahara, Y., Ikeda, A., Maehara, Y., Habara, M., and Toko, K. (2011). Development and evaluation of a miniaturized taste sensor chip. *Sensors*, 11(10):9878–9886.

Tamura, S., Iwano, K., and Furui, S. (2004). Multi-modal speech recognition using optical-flow analysis for lip images. In Wang, J.-F., Furui, S., and Juang, B.-H., editors, *Real World Speech Processing*, pages 43–50. Springer US.

Toyama, T., Sonntag, D., Dengel, A., Matsuda, T., Iwamura, M., and Kise, K. (2014). A mixed reality head-mounted text translation system using eye gaze input. In *Proceedings of the 19th International Conference on Intelligent User Interfaces*, IUI '14, pages 329–334, New York, NY, USA. ACM.

Traum, D. and Larsson, S. (2003). The information state approach to dialogue management. In *Current and New Directions in Discourse and Dialogue*, volume 22 of *Text, Speech and Language Technology*, pages 325–353. Springer Netherlands.

Turk, M. (2014). Multimodal interaction: A review. *Pattern Recognition Letters*, 36:189–195.

Turk, M. and Kölsch, M. (2003). Perceptual interfaces. Technical report, University of California, Santa Barbara.

Vernier, F. and Nigay, L. (2001). A framework for the combination and characterization of output modalities. In Palanque, P. and Paternò, F., editors, *Interactive Systems Design, Specification, and Verification*, volume 1946 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin Heidelberg.

Vilhjálmsson, H. (2009). Representing communicative function and behavior in multimodal communication. In Esposito, A., Hussain, A., Marinaro, M., and Martone, R., editors, *Multimodal Signals: Cognitive and Algorithmic Issues*, volume 5398 of *Lecture Notes in Computer Science*, pages 47–59. Springer Berlin Heidelberg.

Wachs, J. P. (2010). Gaze, posture and gesture recognition to minimize focus shifts for intelligent operating rooms in a collaborative support system. *International Journal of Computers, Communications & Control*, 5(1).

Wachs, J. P., Kölsch, M., Stern, H., and Edan, Y. (2011). Vision-based hand-gesture applications. *Communications ACM*, 54(2):60–71.

Wahlster, W. (1991). User and discourse models for multimodal communication. In J. W. Sullivan and S. W. Tyler, editor, *Intelligent User Interfaces*, pages 45–67. ACM Press.

Wahlster, W. (1992). An intelligent multimodal interface. In Buchmann, J., Ganzinger, H., and Paul, W., editors, *Informatik*, volume 1 of *TEUBNER-TEXTE zur Informatik*, pages 481–494. Vieweg+Teubner Verlag.

Wahlster, W. (2002). 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. MIT Press.

Wahlster, W. (2003). SmartKom: Symmetric multimodality in an adaptive and reusable dialogue shell. In Krahl, R. Günther, D., editor, *Proceedings of the Human Computer Interaction Status Conference 2003*, pages 47–62. DLR.

Wahlster, W. (2006a). Dialogue systems go multimodal: The SmartKom experience. In Wahlster (2006b), pages 3–27.

Wahlster, W., editor (2006b). *SmartKom: Foundations of Multimodal Dialogue Systems.* Springer, Berlin.

Wahlster, W., editor (2013). *SemProM - Foundations of Semantic Product Memories for the Internet of Things.* Cognitive Technologies. Springer.

Wahlster, W., Grallert, H.-J., Wess, S., Friedrich, H., and Widenka, T., editors (2014). *Towards the Internet of Services: The THESEUS Program.* Springer, Berlin.

Wang, K. (2002). SALT: a spoken language interface for web-based multimodal dialog systems. In *Proceedings of ICSLP—Interspeech 2002: 7th International Conference on Spoken Language Processing, Denver, CO, USA*, pages 2241–2244.

Wasinger, R. (2006). *Multimodal interaction with mobile devices: Fusing a broad spectrum of modality combinations.* Aka Verlag.

Wasinger, R., Kray, C., and Endres, C. (2003). Controlling multiple devices. In *Physical Interaction (PI03) Workshop on Real World User Interfaces*, pages 60–63.

Wasinger, R., Krüger, A., and Jacobs, O. (2005). Integrating intra and extra gestures into a mobile and multimodal shopping assistant. In Gellersen, H.-W., Want, R., and Schmidt, A., editors, *Pervasive Computing*, volume 3468 of *Lecture Notes in Computer Science*, pages 297–314. Springer Berlin Heidelberg.

Wütherich, G., Hartmann, N., Kolb, B., and Lübken, M. (2009). *Die OSGi Service Platform.* dpunkt.verlag.

Xiao, B., Girand, C., and Oviatt, S. L. (2002). Multimodal integration patterns in children. In Hansen, J. H. L. and Pellom, B. L., editors, *INTERSPEECH.* ISCA.

Xiao, B., Lunsford, R., Coulston, R., Wesson, M., and Oviatt, S. (2003). Modeling multimodal integration patterns and performance in seniors: Toward adaptive processing of individual differences. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, ICMI '03, pages 265–272, New York, NY, USA. ACM.

Xiao, Y., Zhang, Z., Beck, A., Yuan, J., and Thalmann, D. (2014). Human-robot interaction by understanding upper body gestures. *Presence*, 23(2):133–154.