Universität Rostock Traditio et Innovatio

Device Cooperation in Ad-hoc Multimedia Ensembles

Dissertation

to Obtain the Academic Degree of

Doktor-Ingenieur (Dr.-Ing.)

of the Faculty of Computer Science and Electrical Engineering

at the University of Rostock

Submitted by

Christiane Plociennik

urn:nbn:de:gbv:28-diss2011-0161-5

Reviewers:
Prof. Dr. Thomas Kirste, University of Rostock
Prof. Dr. Adelinde Uhrmacher, University of Rostock
Prof. Dr. Hartmut Wandke, Humboldt University Berlin

Date of Submission: January 12, 2011
Date of Defense: October 11, 2011

Device Cooperation in Ad-hoc Multimedia Ensembles

Abstract: Today's environments are often equipped with a complex device infrastructure. People that use such environments can be overwhelmed by the abundance of functionality such device ensembles offer. Therefore, it is beneficial to assist users in operating the devices. This is what *smart environments* do. A special kind of smart environments are so-called *ad-hoc environments*. Here, the device infrastructure is not static. Instead, it includes mobile devices like PDAs or notebooks that may join and leave the device ensemble at run-time. Nevertheless, users expect the devices to act as one coherent ensemble. The focus of this thesis lies in the question how assistance in such smart ad-hoc environments can be realized. By analyzing a set of smart environment scenarios, we elicit a number of requirements that the assistance should meet; the most important requirement being distributedness. We then present an approach that fulfills these requirements: Under the assumption that the user's goals are known to the ensemble, it enables the devices in an ad-hoc environment to cooperatively generate and execute an action sequence to fulfill these goals. This approach is based on declarative descriptions of the devices' actions and is completely distributed. Because it uses only local knowledge, suboptimal action sequences can occur. In a quantitative user study, we show that users accept such suboptimal assistance if they perceive that it offers them sufficient benefit over manual control of the devices.

Keywords: Strategy Synthesis, Action Selection, User Acceptance, Ubiquitous Computing

Spontane Kooperation von Multimedia-Geräten

Zusammenfassung: Moderne Umgebungen sind oft mit einer komplexen Infrastruktur an technischen Geräten ausgestattet. Menschen, die solche Umgebungen benutzen, sind zunehmend überwältigt von der Fülle an Funktionalität, die solche Geräteensembles bieten. Deshalb ist es sinnvoll, Nutzern solcher Umgebungen bei der Bedienung der Geräte zu assistieren. Umgebungen, die solche Unterstützung anbieten, heißen *Smart Environments*. Eine Spezialform der Smart Environments sind die so genannten *Ad-hoc-Umgebungen*. Sie zeichnen sich dadurch aus, dass die Geräteinfrastruktur nicht statisch ist: Neben fest installierten Geräten gibt es hier auch mobile Geräte wie PDAs oder Notebooks, die dem Ensemble zur Laufzeit hinzugefügt oder auch wieder entfernt werden können. Trotz dessen erwartet der Nutzer, dass alle Geräte als ein kohärentes Ensemble zusammenarbeiten. Der Fokus dieser Arbeit liegt auf der Frage, wie die Assistenz in solchen intelligenten Ad-hoc-Umgebungen realisiert werden kann. Mithilfe von Smart-Environment-Szenarien identifizieren wir Herausforderungen, die von der Assistenz erfüllt werden sollten. Die wichtigste Herausforderung ist hierbei die Verteiltheit. Wir präsentieren einen Ansatz, der die Herausforderungen erfüllt: Unter der Annahme, dass die Ziele des Nutzers bekannt sind, versetzt dieser Ansatz das Geräteensemble in die Lage, gemeinsam eine Aktionssequenz zu finden und auszuführen, die die Nutzerziele erfüllt. Der Ansatz basiert auf deklarativen Beschreibungen der möglichen Geräteaktionen und arbeitet komplett verteilt. Da er nur lokales Wissen verwendet, können suboptimale Aktionssequenzen auftreten. In einer quantitativen Nutzerstudie weisen wir nach, dass Nutzer auch solche suboptimale Assistenz akzeptieren, wenn sie der Meinung sind, dass die Assistenz genügend Vorteile gegenüber der manuellen Bedienung der Geräte bietet.

Schlagworte: Strategiesynthese, Aktionsauswahl, Nutzerakzeptanz, Ubiquitous Computing

Contents

List of Figures

List of Abbreviations

AdDCo algorithm	Ad-hoc Device Cooperation algorithm
AGV	automatic guided vehicle
ANN	artificial neural network
ANOVA	analysis of variance
ASMsg	action selection message
BBL	behavior-based layer
BPEL4WS	Business Process Execution Language for Web Services
CDPS	Cooperative Distributed Problem Solving
CMI	CompModInst
CPL	cooperative planning layer
DAML-S	DARPA agent markup language for services
DSD	DIANE Service Description
ECo	Ensemble Communication Framework
EIB	European Installation Bus
HTN	hierarchical task network
LPL	local planning layer
NMMsg	network maintenance message
OPOSSum	Online Portal for Semantic Services
OWL-S	Web Ontology Language for Services
PDDL	Planning Domain Definition Language
PE	perceived ease of use

PM algorithm	Pattie Maes' algorithm
PTAM	Technology Acceptance Model for Pervasive Computing
PU	perceived usefulness
SAWSDL	Semantic Annotations for WSDL
SWRL	Semantic Web Rule Language
TA-EG	Technikaffinität Elektronische Geräte; english: technophilia with respect to electronic devices
TAM	Technology Acceptance Model
UC-AM	Ubiquitous Computing Acceptance Model
UPnP	Universal Plug and Play
UTAUT	Unified Theory of Acceptance and Use of Technology
WSDL	Web Services Description Language
WSDL-S	Web Service Semantics
WSMO	Web Service Modeling Ontology
XML	Extensible Markup Language

CHAPTER 1

Introduction

Contents

1.1 What Are Smart Environments?

The world we live in is becoming increasingly complex. It is becoming ever more common for us to use technical devices in different parts of our everyday lives. Today many people own a desktop computer, a laptop, a cell phone and a PDA. Meeting rooms often contain computers, projectors, motor canvasses, and sun blinds. People use navigation systems in their cars and entertainment systems in their homes.

With the increasing number and diversity of devices the question how to control all those devices becomes more and more important. We often want to do things that involve a combination of devices. To watch a DVD movie, for example, we have to turn on the DVD player, turn on the TV screen, switch to the video channel, insert the DVD into the player and press *Play*. Many people are confused by the abundance of functionality modern devices offer and wonder how to control all those devices.

It is therefore necessary that computer scientists investigate how to assist people in such technology-rich environments. The branch of computer science concerned with this question is called *Ubiquitous Computing*, *Pervasive Computing*, *Smart Environments*, *Intelligent Environments* or *Calm Computing*. Especially in Europe it is also referred to by the term *Ambient Intelligence*. Ubiquitous Computing research is centered around Weiser's vision [Weiser 1991]:

> "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

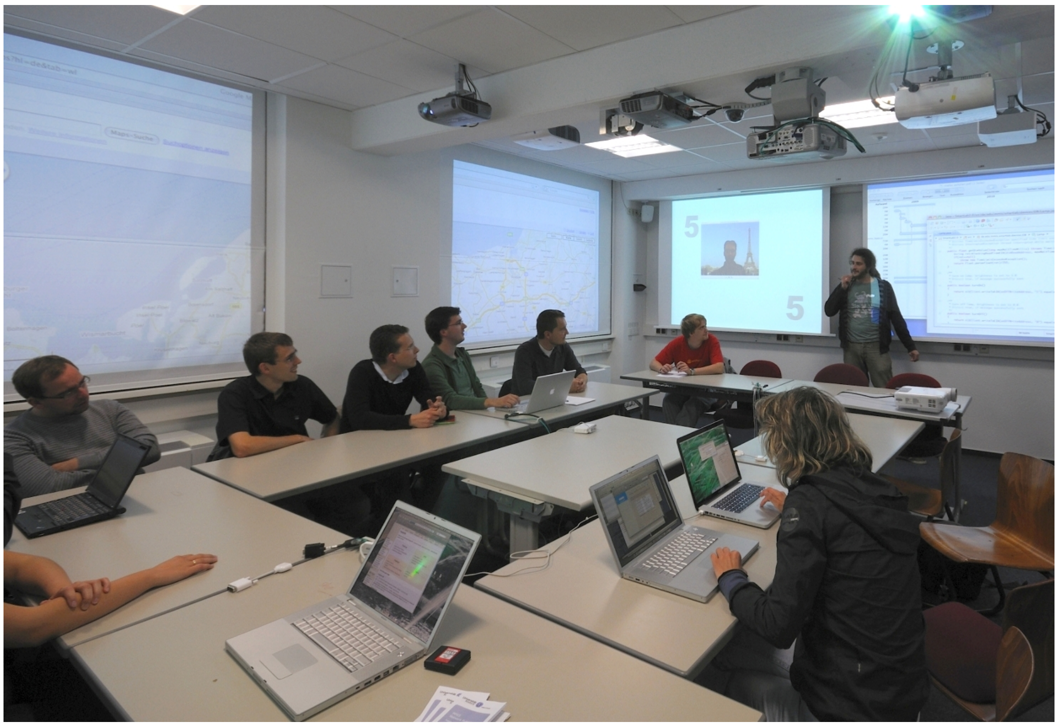


Figure 1.1: A typical usage situation in a smart environment.

But why is it desirable that technology disappears? To answer this question, let us look at the way this kind of technology is used. It differs in many aspects from the way people have used technology – and especially computing devices – a few decades ago. Twenty years ago, people used to sit in front of a desktop computer and devoted all their attention to this computer. Today, most people own a number of smaller, mobile computing devices like laptops, mobile phones or PDAs. Computers are embedded in everyday devices like DVD players. This has changed the circumstances under which people use technology: Users are often situated in "real life" and have a primary task other than operating the devices. For example, if a speaker is about to give a presentation in a meeting or at a conference in front of a huge audience, s/he will have other things on her/his mind (i.e. the upcoming presentation) than figuring out how to control the equipment in the conference room (see Figure 1.1). For this reason, it is beneficial that technology assists the user unobtrusively and does not demand the user's full attention.

It is a huge challenge to make Weiser's vision come true for environments with a fixed device infrastructure. Yet it is an even bigger challenge to achieve the same in so-called *ad-hoc environments* which we examine in this thesis. Ad-hoc environments do not consist of a fixed device ensemble. Consider an example smart meeting room: It contains several projectors, several canvasses, some sunblinds, and a set of dimmable lights. In addition, users bring their own mobile devices like laptops or PDAs into the environment. Those mobile devices should integrate with the fixed devices to form an ad-hoc ensemble (see Figure 1.2). The most radical form of an ad-hoc ensemble is the so-called *white room scenario*: Such a room contains no fixed devices apart from a network infrastructure; all devices are brought by the users.

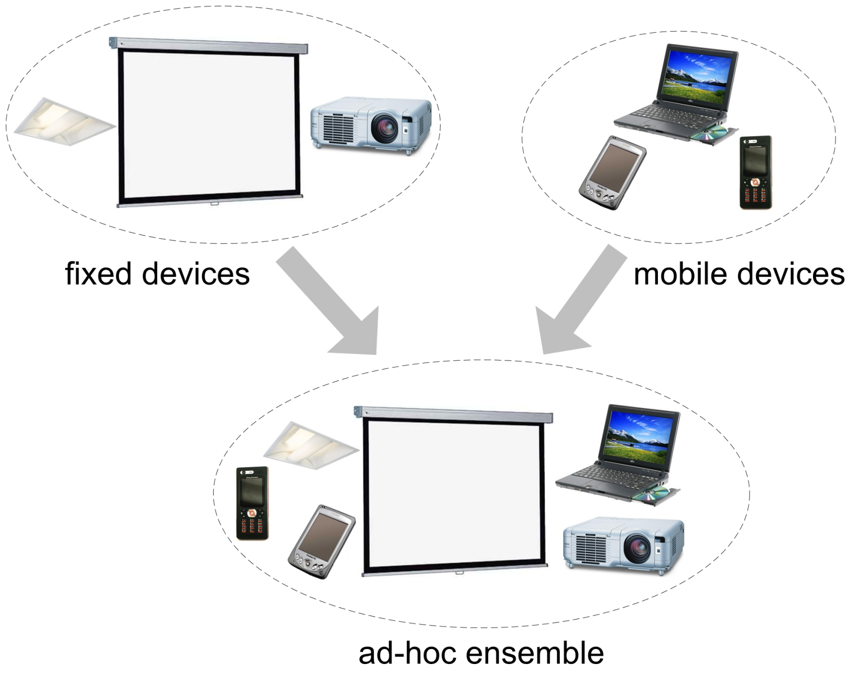


Figure 1.2: An ad-hoc ensemble in a smart meeting room.

1.2 What Is User Assistance?

User assistance can be divided into two stages: the *intention analysis* and the *strategy synthesis* (see Figure 1.3). The intention analysis tries to find out what the user wants the ensemble to do for her/him. Typically, machine learning approaches are used for inferring such user goals from sensor data and a priori knowledge about the user's behavior. In our group, we experiment with inferring the goals of a team of users based on a dynamic Bayesian model of team behavior [Giersich & Kirste 2007]. The inferred goals can then serve as the basis for the strategy synthesis, which tries to help the human achieve her/his goals. Often, this involves composing a sequence of device actions which, when executed, will fulfill the goals. The main topic of this thesis is to investigate which kind of strategy synthesis mechanism is appropriate for smart ad-hoc environments. Hence, we assume that the intention analysis works correctly and provides the user goals.

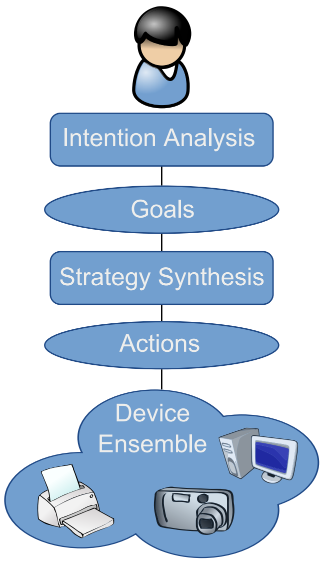
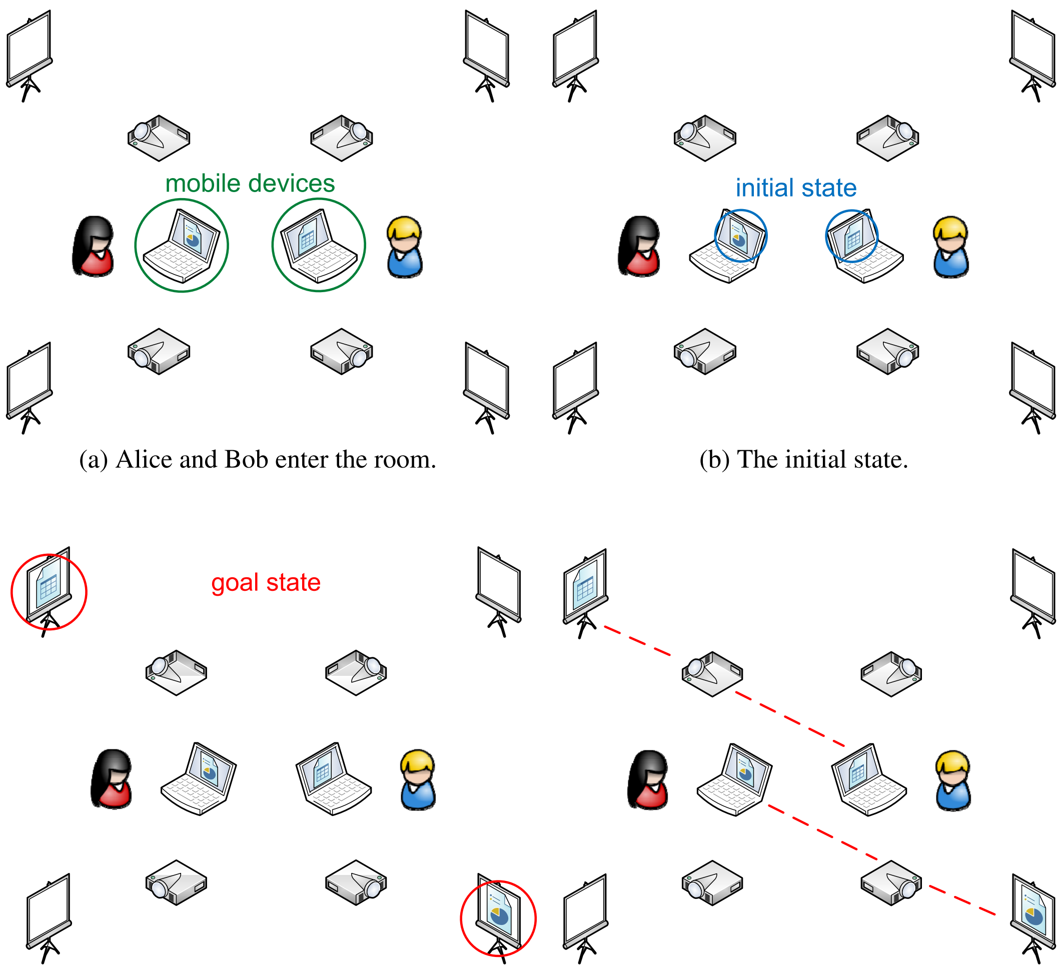


Figure 1.3: User assistance in two stages.

To exemplify the process of user assistance, we take a look at a very simple scenario from the smart meeting room domain:[1] Alice and Bob meet to discuss the next steps for a project they are both involved in. The meeting takes place in a room instrumented with four projectors and four canvasses. Alice and Bob each bring a notebook (see Figure 1.4a). They have both prepared a set of slides they want to show, which they now open and maximize on their notebook screens (see Figure 1.4b). We call this the *initial state*. It can be sensed by the environment by a number of hardware and software sensors. In our example, the sensors might be software daemons that run on Bob's and Alice's notebooks and that notify the smart environment when a powerpoint presentation is started. Bob wants to show his presentation on the upper left canvas; Alice wants to show hers on the lower right canvas (see Figure 1.4c). These goals must be inferred by the intention analysis. Once the intention analysis has informed the strategy synthesis about the goals, the

[1]More scenarios follow in Chapter 2.

strategy synthesis can find the following action sequence that fulfills them: Bob's notebook sends Bob's presentation to one of the projectors, which then projects it onto the upper left canvas. Alice's notebook sends Alice's presentation to another projector, which projects it onto the lower right canvas. Depending on the level of automation an assistance system exhibits, the strategy synthesis can be performed entirely by the human, partly by the human and partly by the system, or entirely by the system. This is discussed in the following section.



(a) Alice and Bob enter the room.

(b) The initial state.

(c) The goal state.

(d) The action sequence generated.

Figure 1.4: User assistance in ad-hoc environments.

1.3 Which Assistance Does The User Want?

The focus of this thesis is on the technical side of strategy synthesis. However, one cannot investigate *which mechanisms* are suited for assisting people without understanding *how much* and *which kind* of assistance people want and accept. This question is related to the question how much automation an operator of a machine wants, which has been investigated in the field of automation science. We therefore take some time to introduce different frameworks of user involvement.

Sheridan proposes an eight-level scale of automation (cf. [Sheridan 2002]) – from *no assistance at all* where the human selects and executes the method to do the task without any help by the computer to *full automation* where the computer selects and executes the method to do the task autonomously, without any human involvement:

1. The computer provides no assistance; the human must do the task completely her-/himself.
2. The computer offers a selection of alternatives to do the task.
3. The computer chooses how to do the task and
4. executes the task in this way if the human approves, or
5. allows the human to veto within a fixed timespan before executing the task automatically, or
6. executes the task automatically, then informs the human, or
7. executes the task automatically, then informs the human only if asked.
8. The computer chooses the method, executes the task, and ignores the human.

In fact, several revisions of this scale with different numbers of stages exist. This is the most recent version. Sheridan's model has gained wide acceptance in the research community (see e.g. [Proud *et al.* 2003], [Cummings *et al.* 2007]). However, Wandke makes the following remarks about Sheridan's scale [Wandke 2005]:

- Sheridan's taxonomy refers only to decision-making and action implementation and does not consider different action stages.
- There are simpler kinds of assistance that do not require a decision to be taken. For such cases, Sheridan's eight stages might be too sophisticated.
- Sheridan's model is tailored to assistance in operating situations, where spe-

cialized operators control machines and intervene in case an error occurs. When it comes to using interactive devices rather than autonomous processes, the model must be extended.

As a result of this discussion, Wandke proposes a conceptual framework for assistance systems [Wandke 2005]. From his point of view, assistance systems strive to make functions of a system accessible to the user which would otherwise be inaccessible. Wandke gives the following possible reasons for inaccessibility of functions:

- The user is unaware that the function exists.
- The user does not know how to apply the function.
- The user finds that applying the function her-/himself is too difficult because it requires too much sensory, cognitive, or motor effort.

All of these reasons may lead to the user not using the system at all, or using it ineffectively. Thus, Wandke sees assistance as a bridge between the human's capabilities and system functionality. We adopt his framework for our own work as it is more comprehensive than Sheridan's model. Furthermore, while Sheridan developed his model with operational contexts in mind, Wandke's framework is tailored to assistance systems. Assistance systems can be found in such diverse fields as driving assistance in cars, smart homes, or speech output systems for blind users. Yet they all have in common that their users are usually non-experts in day-to-day situations. This framework is thus better suited for the domain of Ubiquitous Computing than Sheridan's model. Wandke classifies assistance according to three dimensions:

- the stages of human-machine interactions that can be assisted
- adjustment
- initiative

In the following, we explain these three dimensions. We start with the *stages of human-machine interactions* that can be assisted by technical components. Wandke distinguishes six of them. For the sake of brevity, we introduce them here in a concise fashion. The examples are also taken from [Wandke 2005]:

1. **Motivation, activation, and goal setting**: A machine helps the user establish the right goals, or it motivates or activates the human. For instance, an

assistance system can help reinforce the user's motive or cause the user to continue with an ongoing activity. An example is the message "Please hold the line" which users hear on busy phone lines.

2. **Perception**: In this stage, the machine helps the user perceive sensor data by e.g. amplifying signals or transforming them into a different mode. An example is displaying traffic signs on the car dashboard.
3. **Information integration, generating situation awareness**: Here, the machine helps the user interpret information. An example is a tool-tip that explains the function of a button in a program when the user moves the mouse onto the button.
4. **Decision making, action selection**: The machine helps the user select and possibly execute the appropriate actions. An example is a *wizard* that guides a user through the installation process of a program.
5. **Action execution**: A machine helps the user or operator execute actions, e.g. because the human is physically too weak. The user is, however, always in control. An example is the brake assistance in driving.
6. **Processing feedback of action results**: In this stage, a machine helps the user realize or interpret the results of her/his actions. An example is a parking assistant that emits a sound sequence depending on the distance to a neighboring car.

Assistance systems usually assist the user in one or a few, but seldom in all of these stages. Our research is concerned with the *decision making/action selection step*, hence we now examine this stage in more depth. Wandke proposes a seven-level taxonomy for decision making and action selection which is influenced by Sheridan's eight levels:

- **Supply assistance**: The system presents all currently available action options to the user, without any filtering. An example is a handbook that offers a description of the user's action options depending on the current state. The problem with this level is that the user may be overwhelmed by the abundance of options.
- **Filter assistance**: Of all currently available action options, the system selects some and presents them to the user. Actions are selected according to some filter mechanism, e.g. the likelihood of success if the user chooses to execute

this action. An example is a *wizard* that guides a user through the installation process of a program.

- **Adviser assistance**: The system presents only one action option to the user, who may or may not follow the recommendation. An example is a navigation system in a car which presents only one possible route to the destination. The problem with this level is that it may not be accepted in open situations because users perceive it as annoying.
- **Delegation assistance**: The system presents one action option to the user and executes this action automatically if the user agrees. In case the user rejects the proposal, the system may recommend another action or wait for more information. An example is a car entertainment system that recommends music to the user and starts replaying it if the user agrees.
- **Take-over assistance**: The system proposes an action to the user and executes it automatically if the user does not refuse the action within a limited amount of time. An example are systems that detect when the user is not monitoring: If the operator does not respond to a signal, a security function is executed.
- **Informative execution assistance**: All actions are executed by the system automatically. The user supervises the system and can switch off the assistance in case of failure. An example is the autopilot.
- **Silent execution assistance**: The system executes all actions automatically without informing the user. An example is automatic gear changing. If the system is highly reliable, this kind of assistance can be very convenient. However, if this is not the case, the user may have to take over suddenly in case of failure without being prepared.

Next, we describe the possibilities for *adjustment*. This refers to the question how the appropriate type of assistance should be selected. Wandke's framework distinguishes four possibilities for adjustment:

- **Fixed assistance**: The same assistance system is used by all users in all situations. An example is the electronic brake assistant in a car.
- **Customized assistance**: Such systems are custom-tailored to the needs of specific groups, or customized for specific tasks, or tailored to specific contexts. Customization takes place during the design phase of the system.

- **Adaptable assistance**: The user can adjust the system to fit her/his needs, tasks, or the context. An example is the gearbox of a car: One driver may select automatic gears, another driver may choose a Tiptronic function.
- **Adaptive assistance**: The system itself adapts the type of assistance it provides according to the current context or stored features. An example is a driving assistant that presents instructions to the user via textual output or speech, depending on whether the user is looking at a screen or elsewhere.

The last dimension to look at is *initiative*. Wandke's framework distinguishes two types of initiative:

- **Passive assistance**: Assistance is initiated by the user. This has the advantage that the user is always the master of the situation. However, the user must know that s/he is in need of assistance, that there is an assistance system capable of this task, and how to get the system to provide the right assistance in the current context. Furthermore, s/he must have enough free time and cognitive resources to initiate assistance.
- **Active assistance**: This type of assistance is sometimes also called *proactive*. Assistance is initiated by the system based on the user's preferences or certain features of the current context. Active assistance is preferable if the prerequisites for passive assistance are not met, or if fast task completion is necessary (e.g. in safety-critical situations). However, active assistance runs the risk that users may feel patronized.

The question is now, of course, where in this framework the perfect assistance system for smart environments fits in. Each of the three dimensions we just introduced has to be taken into account: Which *level of decision making/action selection* is appropriate for our domain? What kind of *adjustment* is preferable? And in terms of *initiative*, does the user prefer passive or active assistance? In order to answer these questions, we perform an analysis of the domain of smart environments in Chapter 2: We look at a variety of scenarios other researchers have published. This literature review will guide us in forming a hypothesis which kind of assistance the user in our domain wants. The hypothesis is then tested in a user study in Chapter 6.

1.4 Outline and Contributions of this Thesis

Having settled the necessary preliminaries, we can now present the structure of this thesis and elaborate on its contributions.

In Chapter 2 we perform a comprehensive analysis of the domain of smart environments with the help of scenarios from the literature. Our goal is to identify requirements a strategy synthesis mechanism in smart environments has to fulfill. Furthermore, this domain analysis will enable us to form a hypothesis which kind of assistance in terms of Wandke's assistance framework users in smart environments prefer.

In Chapter 3, a review of the state of the art in several fields relevant to smart environments is given. We believe that the question whether deliberative or reactive control is more appropriate for the strategy synthesis in smart environments is a fundamental one. To be more precise, a key to finding a successful mechanism for strategy synthesis lies in determining the appropriate point in the continuum of deliberative and reactive control. Therefore, the research projects introduced in Section 3 are all classified with respect to this dichotomy.

In Chapter 4, an appropriate mechanism for strategy synthesis is selected based on the requirements identified in Chapter 2 and on the discussion in Chapter 3. We chose to distribute and adapt an approach proposed by Maes [Maes 1990b]. The resulting algorithm, called the AdDCo algorithm, is described in Chapter 4. The last section of Chapter 4 classifies the AdDCo architecture with respect to two agent models. In this context, a new kind of architecture is introduced: the temporally layered architecture.

Chapter 5 introduces some more enhancements that cause the AdDCo algorithm to better fulfill the requirements introduced in Section 2.2: We describe why extending the expressivity of effects to incorporate universal quantification improves *flexibility*. We furthermore elaborate on the question how *persistent actions* can be accounted for by the strategy synthesis in ad-hoc environments. We compare the approach one can take when using planning as a strategy synthesis mechanism to the approach possible when using the AdDCo algorithm. Moreover, we describe how the AdDCo algorithm can be easily extended to further enhance *design-time modularization*.

The AdDCo algorithm is a distributed algorithm; there is no component that has global knowledge. Due to this, the action sequences generated by the algorithm are often suboptimal. An interesting question is how users react to such sequences. In Chapter 6, we therefore investigate whether users accept suboptimal assistance and under which circumstances they accept it. The assistance system used in this study is based on the AdDCo algorithm. We investigate how three factors influence user acceptance: task load, experience, and system behavior. In this user study we furthermore investigate whether the hypothesis regarding Wandke's assistance framework from Chapter 2 is correct.

The AdDCo algorithm generates strategies at run-time based on declarative descriptions of the actions of the devices in the ensemble. These descriptions reside directly on the devices. As the devices are likely to stem from different vendors, their descriptions are likely to stem from different developers, too. For the devices to be able to form one coherent ensemble, the descriptions must nevertheless be written in one common language. Otherwise, it would not be possible for the devices to communicate about their actions. An interesting question is which languages can be used for this purpose. We investigate this in Section 7, focusing on semantic web service languages.

In Section 8 we conclude this thesis and give an outlook.

In a nutshell, this thesis makes the following contributions:

- We identify requirements that must be met by a successful strategy synthesis mechanism in smart environments (Chapter 2).
- We discuss how an assistance system for smart environments should fit into Wandke's assistance framework (Chapters 2, 6).
- We review state-of-the-art research (Chapter 3) and classify the research projects with respect to the continuum of deliberative and reactive control. Part of this work has been published in [Reisse *et al.* 2007].
- We describe a distributed strategy synthesis mechanism for smart ad-hoc environments which we call the AdDCo algorithm (Chapters 4, 5). This work has been published in [Reisse & Kirste 2008b, Reisse & Kirste 2008a].
- We introduce a new kind of layered architecture: the temporally layered architecture (Chapter 4). The AdDCo algorithm is based on this architecture.
- We describe why supporting universally quantified effects changes the modelling of actions in smart environments such that flexibility is improved. This

work has been published in [Reisse & Kirste 2008b, Reisse & Kirste 2008a].

- We discuss how the persistent action problem can be solved. This has been published in [Plociennik *et al.* 2009].
- We present the results of a quantitative user study that investigates whether users accept suboptimal assistance, and which contextual factors influence this acceptance (Chapter 6). The factors considered are task load, experience, and system behavior. This work has been published in [Plociennik *et al.* 2010].
- We investigate which semantic web service languages are suited for describing device actions declaratively, which is needed for distributed strategy synthesis mechanisms like the AdDCo algorithm (Chapter 7). This work has been published in [Reisse *et al.* 2008a, Reisse *et al.* 2008b].

CHAPTER 2

Smart Environments – A Domain Analysis

Contents

Before starting to look for a suitable strategy synthesis mechanism, one must elicit a set of requirements this strategy synthesis mechanism has to meet. To identify the requirements, it is essential to have a good understanding of possible use cases of assistance systems in our domain. The problem is that few “real” smart environments exist today. Most are prototypes in research institutes. Therefore, it is not possible to observe users while they are using such environments. Another way to pinpoint a wide variety of different use cases is to look at smart environment scenarios from the literature, which we do in Section 2.1. This enables us to elicit a set of requirements on strategy synthesis in Section 2.2, and to develop a hypothesis which kind of assistance in terms of Wandke’s assistance framework is appropriate in smart environments in Section 2.3. In Section 2.4, we explain why such an assistance system can be regarded as a multi-agent system.

2.1 Survey of Smart Environment Scenarios

In the following, we analyze 15 smart environment scenarios published in the last ten years.[1] We briefly describe each scenario. As some of the original scenarios are quite long we take the liberty to shorten them in an appropriate way. Each scenario description is followed by a classification of the assistance provided in the scenario according to Wandke’s assistance framework. We then state whether it is an ad-hoc scenario and if an action sequence is generated by the system. What this means is explained in Section 2.2. If this is the case, we present an example action sequence.[2] We also state each action sequence’s length. This is important to get an idea of the complexity of the problems that need to be solved in typical smart environments. We furthermore present any special requirements this scenario poses on an assistance system.

[1]The analysis of the last three scenarios (Fujii and Suda, Aura, and Daidalos) is based on [Marquardt & Uhrmacher 2009].

[2]We wrote most of the example action sequences ourselves because many authors do not include such examples in their scenario descriptions.

2.1.1 The Ozone Project [Issarny *et al.* 2005]

- **Scenario:** Four friends meet in the city of Rocquencourt for a tennis match which they have scheduled in advance using their individual agendas. Two of them, John and Paul, arrive by train from Paris. Having arrived at Rocquencourt train station, they reserve seats in an automated vehicle called a CyberCar which takes them to the tennis court. On the way they call their friend Michel via John's PDA and ask him if he can recommend a movie for the evening. Michel sends a trailer via his PDA which John and Paul watch. There is a lot of noise in the street and thus the audio content is not audible. Thus, textual output is presented to John and Paul. They agree on a movie and ask Sylvie, their other friend. Afterwards, they purchase the tickets. They organize the rest of the day (dinner, transport) in the same fashion.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Supply assistance.
  - *Adjustment:* Adaptive assistance.
  - *Initiative:* Passive assistance.
- **Is it an ad-hoc scenario?** Yes.
- **Must an action sequence be generated?** No. The system just provides seamless integration of services on different devices. The services are invoked by the user.
- **Example action sequence:** –
- **Number of actions in this sequence:** –
- **Special requirements of this scenario:** –

2.1.2 The Intelligent Room [Coen 1997]

- **Scenario:** A user stands in front of a scalable map of the Carribean. S/he points at Puerto Rico and requests the computer to zoom in by uttering the command "Computer, zoom in". S/he then points at San Juan and once again orders the computer via speech to zoom in. Finally, s/he asks the computer via speech about the weather there.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Supply assistance.
  - *Adjustment:* Fixed assistance.

  - *Initiative:* Passive assistance.
- **Is it an ad-hoc scenario?** No.
- **Must an action sequence be generated?** No, all actions are invoked by the user by uttering commands preceded by the word "computer".
- **Example action sequence:** –
- **Number of actions in this sequence:** –
- **Special requirements of this scenario:** –

2.1.3 The Amigo project [Mokhtar *et al.* 2005]

- **Scenario:** Two boys, Robert and his friend, meet at Robert's house and want to watch a movie from the home movie database using the portable DVD player Robert's friend brought with him. The DVD player checks the access rights, adjusts the luminosity and sound volume according to the user preferences, and then plays the movie.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** Yes.
- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(getFilm film)* $\rightarrow$ *(checkAccessRights film Robert)* $\rightarrow$ *(setLuminosity Robert)* $\rightarrow$ *(setVolume Robert)* $\rightarrow$ *(playFilm film Robert)*
- **Number of actions in this sequence:** 5
- **Special requirements of this scenario:** *playFilm* is a persistent action, which is explained in Section 2.2. Any strategy synthesis mechanism suited for this scenario must be able to cope with persistent actions.

2.1.4 Amigoni et al. [Amigoni *et al.* 2005]

- **Scenario:** The assistance system checks whether insulin is available in a medical store. If no insulin is left, it makes a request to pharmacies to provide insulin. It has various modalities for this request: It can make a phone call, send an SMS, an e-mail or a fax. Each of those possibilities is annotated with numerical values for the parameters *performance*, *cost*, and *probability of success*. These parameters can be used to select the optimal modality with respect to the specific problem instance.

- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** Yes.
- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(CreateRequestText insulin text)* $\rightarrow$ *(SearchEmailAddress insulin address)* $\rightarrow$ *(SendEmail text address)*
- **Number of actions in this sequence:** 3
- **Special requirements of this scenario:** Involves optimization. Furthermore, the scenario requires to decompose tasks into subtasks. Thus, information about possible decompositions (in other words, hierarchical knowledge) must be available, and a central component must decide which decompositions to apply.

2.1.5 MavHome [Das *et al.* 2002]

- **Scenario:** Bob, the inhabitant of the MavHome smart home, usually gets up at 7 am. From past experience the house has learned this fact and turns on the heating at 6:45 am. Bob's alarm clock signals the bedroom light and the coffee maker to go on. Bob walks into the bathroom and turns on the light, which signals MavHome to display morning news on the bathroom video screen and to turn on the shower.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** No.
- **Must an action sequence be generated?** Yes, but each sequence consists of only one action.
- **Example action sequence:** *(turnOnHeating)*
- **Number of actions in this sequence:** 1

- **Special requirements of this scenario:** –

2.1.6 Adaptive Home [Mozer 2005]

- **Scenario:** The inhabitant of a smart home leaves the house at 8 am. The home predicts s/he will return by 6:30 pm because s/he had returned by that time on the previous days. Thus, the house runs the furnace to achieve the appropriate temperature by the time the inhabitant returns. When the inhabitant returns, s/he prepares dinner. The home switches on the lights to full intensity in the kitchen and great room. The inhabitant then relaxes on the couch. The house dims the lights. On a weekend the house predicts that the inhabitant will not return home before midnight and that only the master-bedroom needs to be heated then. Hence, it uses the electric space heaters in the bedroom instead of the whole-house furnace. When the inhabitant climbs out of bed at 4 am, the home predicts that the bathroom will be used. Thus, it turns on bathroom lights with low intensity.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** No.
- **Must an action sequence be generated?** Yes, but each sequence consists of only one action.
- **Example action sequence:** *(turnOnFurnace)*
- **Number of actions in this sequence:** 1
- **Special requirements of this scenario:** –

2.1.7 Misker et al. [Misker *et al.* 2005]

- **Scenario:** Eric wants to show pictures from his mobile phone to Fiona on a larger screen in a café. He connects the tabletop screen, his mobile phone and his photo database at home and controls the slideshow with his phone. While showing the photos, he adds voice comments to the photos using his phone.
  - *Decision making/action selection:* Supply assistance.

    - *Adjustment:* Fixed assistance.
    - *Initiative:* Passive assistance.
- **Is it an ad-hoc scenario?** Yes.
- **Must an action sequence be generated?** No. The system's purpose is to provide seamless integration of services on different devices. The services are invoked by the user.
- **Example action sequence:** –
- **Number of actions in this sequence:** –
- **Special requirements of this scenario:** –

2.1.8 Connelly and Khalil [Connelly & Khalil 2004]

- **Scenario:** When a user walks into an office, her/his mp3 player turns off automatically. Likewise, when the user attends a meeting, the cell phone ringer is turned off.
- **Which kind of assistance in Wandke's framework?**
    - *Decision making/action selection:* Informative execution assistance.
    - *Adjustment:* Fixed assistance.
    - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** Yes.
- **Must an action sequence be generated?** Yes, but each sequence consists of only one action.
- **Example action sequence:** *(turnOffMP3Player)*
- **Number of actions in this sequence:** 1
- **Special requirements of this scenario:** –

2.1.9 Just Play [Paluska *et al.* 2006]

- **Scenario:** A user wants to play a video from her/his laptop computer on the TV and the home entertainment system.
- **Which kind of assistance in Wandke's framework?**
    - *Decision making/action selection:* Informative execution assistance.
    - *Adjustment:* Fixed assistance.
    - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** Yes.

- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(setVideoOutput laptop TV) → (setAudioOutput laptop homeEntertainmentSystem) → (playVideoOnTV laptop file TV)*
- **Number of actions in this sequence:** 3
- **Special requirements of this scenario:** *playVideoOnTV* is a persistent action. This is explained in Section 2.2.

2.1.10 IHome [Lesser *et al.* 1999]

- **Scenario:** Lesser et al. present a very detailed coffee maker scenario: A coffee maker agent is described by its actions or tasks, e.g. *Acquire-Beans*, *Brew-Coffee*. These tasks are arranged in a hierarchy called a TÆMS task structure that shows how actions can be decomposed into subactions. For example, *Make-Coffee* can be decomposed into *Acquire-Ingredients* and *Hot-Coffee*. The TÆMS task structure may also contain different alternatives to perform a task. The task *Acquire-Beans*, for example, can be decomposed into either one of *Use-Frozen-Beans* or *Buy-Beans-From-Starbucks*. These alternatives are annotated with the parameters cost, quality, and duration. This way, the coffee maker agent can optimize the process of making coffee according to its current situation.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** No, the agents are hard-wired.
- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(Get-Hot-Water) → (Use-Frozen-Beans) → (Grind-Beans) → (Mix-And-Filter) → (Brew-Coffee)*
- **Number of actions in this sequence:** 5
- **Special requirements of this scenario:** Involves optimization. Furthermore, the scenario requires to decompose tasks into subtasks. Thus, information about possible decompositions (in other words, hierarchical knowledge) must be available, and a central component must decide which decompositions to apply.

2.1.11 Intelligent Classroom [Franklin 1998]

- **Scenario:** A speaker in an intelligent classroom is filmed by a camera. As s/he walks from the podium over to the chalkboard, the camera zooms out to follow the speaker. When the speaker has arrived at the chalkboard, the lights are adjusted and the camera is set to show the chalkboard.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** No, the room is prewired.
- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(moveCamera podium)* $\rightarrow$ *(zoomInCamera speaker podium)* $\rightarrow$ *(trackPerson speaker podium)* $\rightarrow$ *(zoomOutCamera)* $\rightarrow$ *(moveCamera chalkboard)* $\rightarrow$ *(zoomInCamera speaker chalkboard)* $\rightarrow$ *(trackChalkboard speaker chalkboard)*
- **Number of actions in this sequence:** 7
- **Special requirements of this scenario:** As the speaker moves from the podium to the chalkboard, the sensor data changes from *(At speaker podium)* to *(At speaker chalkboard)*, and the goal changes from *(Tracked speaker)* to *(Tracked chalkboard)*. Furthermore, *trackPerson* and *trackChalkboard* are persistent actions, which is explained in Section 2.2. Thus, a strategy synthesis mechanism that is suited for this scenario must be able to account for changing sensor data and changing goals as well as persistent actions.

2.1.12 The EMBASSI project [Kirste 2000]

- **Scenario:** A user wants light in the room while minimizing energy consumption. Currently three lights are on and the shutter is closed. The system realizes that it is light outside. It therefore turns off the three lights and opens the shutter.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.

- **Is it an ad-hoc scenario?** No.
- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(open-shutter shutter)* → *(turn-off lamp3)* → *(turn-off lamp2)* → *(turn-off lamp1)* → *(light-on)*
- **Number of actions in this sequence:** 5
- **Special requirements of this scenario:** –

2.1.13 Fujii and Suda [Fujii & Suda 2004]

- **Scenario:** Tom has found a new restaurant on the internet. Now he wants to print out the route from his home to this restaurant without having to bother about details like fetching the restaurant's address from the website or accessing a route planner.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** Yes.
- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(getAddress restaurantAddress)* → *(switchOn printer)* → *(getRoute homeAddress restaurantAddress document)* → *(printDocument document printer)*
- **Number of actions in this sequence:** 4
- **Special requirements of this scenario:** –

2.1.14 Aura [Garlan *et al.* 2002]

- **Scenario:** Jane is at the airport and waits for her flight at gate 23. Before boarding the plane, she wants to send some e-mails. Unfortunately, the wi-fi at gate 23 is overloaded, hence she would not be able to finish the upload before her flight. However, the wi-fi connection at gate 15 is better, and Jane has some time left. The Aura system notices that and suggests Jane to go to gate 15. There, Aura connects to the wi-fi and uploads Jane's e-mails. Jane then walks back to gate 23 and boards her plane.
- **Which kind of assistance in Wandke's framework?**

  - *Decision making/action selection:* Informative execution assistance/Adviser assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** Yes.
- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(move gate23 gate15 laptop)* → *(connectWiFi laptop gate15 highBandwidth)* → *(sendMail e-mails laptop highBandwidth)* → *(move gate15 gate23 laptop)* → *(checkIn laptop gate23)*
- **Number of actions in this sequence:** 5
- **Special requirements of this scenario:** The scenario involves actions of a human. For the scenario to be applicable, Jane must move her laptop from gate 23 to gate 15 and back. Hence, when the *move* action is executed, the assistance system must wait until Jane has actually moved before the next action is scheduled. Feedback about whether the *move* action is complete must come from sensors, e.g. a positioning system. Furthermore, as Marquardt et al. have remarked [Marquardt & Uhrmacher 2009], when human actions are involved, the action sequence should be regarded as a suggestion for acting rather than a plan. The human must have the possibility to opt out easily. Therefore, the level of decision making and action selection in this scenario can be regarded as a combination of *informative execution assistance* and *adviser assistance*. Furthermore, *connectWiFi* is a persistent action, which is explained in Section 2.2.

2.1.15 Project Daidalos [Yang *et al.* 2006]

- **Scenario:** Bart is watching the news on TV when his boss calls and asks him to pick up a customer from the airport. As Bart enters his car, the TV starts streaming the buffered news cast to Bart's PDA. The PDA converts the news to an audio stream, which is then accessed and played by the car radio.
- **Which kind of assistance in Wandke's framework?**
  - *Decision making/action selection:* Informative execution assistance.
  - *Adjustment:* Fixed assistance.
  - *Initiative:* Active assistance.
- **Is it an ad-hoc scenario?** Yes.

- **Must an action sequence be generated?** Yes.
- **Example action sequence:** *(connect PDA carRadio)* → *(stopStream TV recordedNews video)* → *(connect TV PDA)* → *(startStream TV recordedNews video)* → *(accessStream TV PDA recordedNews video)* → *(convertData recordedNews video audio PDA)* → *(accessStream PDA carRadio recordedNews audio)*
- **Number of actions in this sequence:** 7
- **Special requirements of this scenario:** *connect*, *startStream*, and *accessStream* are persistent actions. This is explained in Section 2.2.

2.2 Requirements on the Strategy Synthesis

Based on the survey in the previous section, we now discuss the requirements that must be met by the strategy synthesis in smart environments:

- **Spontaneity**: 9 of the 15 scenarios presented in the previous section are ad-hoc scenarios: Users expect the strategy synthesis to function as soon as they have built up the ad-hoc ensemble; they do not have the time or the will to wait for a training phase to end before they can use the system. Thus, devices must be able to form an ensemble even if they have never been used together, and this ensemble must function right away. This implies that any approach that relies on learning is not suited for the majority of the scenarios.
- **Action sequences**: 12 of the 15 scenarios require the strategy synthesis to generate sequences of actions. However, these are usually rather short. In all the scenarios we looked at, the longest action sequence that had to be created had 7 actions. Of course, this does not mean that longer sequences will never occur. However, it allows us to conclude that in the average scenario that can be found in the literature today the strategy synthesis mechanism of choice need not be able to find long, complicated action sequences (as e.g. in the planning problems from the ICAPS planning competition [IPC 2008]). Of course, the number of actions needed to fulfill a goal depends on the granularity used to model the actions. However, in order to keep the strategy synthesis process as simple as possible, it is beneficial to model actions in a coarse-grained fashion. To this end, we regard all actions that can be executed in a bulk by a single device as *atomic*. Such actions may well be

split into subactions by the respective device when executed, but as no dependencies exist between those subactions and actions of other devices, it is not necessary to consider them as separate actions during the strategy synthesis. Therefore, the lengths of the action sequences in the scenarios can be regarded as representative.

- **Rationality**: This requirement is tightly coupled to the requirement *action sequences*. In all scenarios that require the strategy synthesis to generate action sequences, which applies to 12 of our 15 scenarios, users expect the system to exhibit rational behavior: Of all the possible actions that the system is capable of, the user expects the system to select and execute those that fulfill her/his goals. Any action executed by the system that has nothing to do with the goals may irritate or even annoy the user. Hence, *rationality* is an implicit requirement. As an example, consider the scenario in Section 2.1.11: If the speaker begins her/his talk on the podium and the camera turns to the audience instead of tracking the speaker, this can lead to confusion. We refine the notion of (rational) system behavior in Chapter 6, where we introduce the *four levels of imperfection*.
- **Flexibility**: The strategy synthesis must be able to account for dynamic ensemble structures. In ad-hoc ensembles, which account for 9 of the 15 scenarios, devices join or leave at run-time. Consider for example the scenario in Section 2.1.3: Robert's friend brings along a DVD player which integrates with the device ensemble already present in the home. Thus, any strategy synthesis mechanism that relies on action sequences precompiled by the system designer is not feasible for the majority of the scenarios. Action sequences must be generated at run-time, taking into account the devices currently in the ensemble and their capabilities.
- **Robustness**: Ad-hoc ensembles consist of loosely coupled elements. The devices and their functionality have not been carefully designed to work together. Furthermore, they are possibly connected through a wireless network (see e.g. the scenario in Section 2.1.14). This implies that failures occur more often than in environments with a fixed concerted device infrastructure. Thus, occasionally devices may leave the ensemble unexpectedly, e.g. because network connectivity has decreased immediately. This must not cause a failure of the entire system. The rest of the ensemble must be able to

recover.

- **Support for persistent actions**: Another characteristic of the domain are persistent actions. These occur in 5 of the 15 scenarios. Persistent actions are actions that persist over a longer timespan, in contrast to very short actions. An example for a persistent action is the *project* action of a projector: Once started, it persists until someone terminates it, e.g. by turning off the projector. In contrast, an example for a short action is *turning on a light*.[3] This has practical implications as e.g. a resource such as a projector is occupied as long as it is projecting. Thus, the strategy synthesis mechanism must be able to account for this.
- **Distributedness**: We consider *distributedness* the most important requirement for the following reasons: As can be seen in the scenarios discussed above, ad-hoc ensembles consist of several loosely coupled nodes. Thus, their *hardware* is already distributed, and the *software* that controls this hardware lends itself to a distributed approach, too: Any new device that enters the ensemble can contribute new information and new capabilities to the strategy synthesis. For instance, each device can contribute declarative descriptions of its possible actions, as has been suggested in [Heider 2010]. Furthermore, avoiding a centralized controlling component means avoiding a single point of failure. Plus, as ad-hoc ensembles may consist entirely of resource-constrained devices like PDAs or mobile phones (consider, for example, the scenario in Section 2.1.15), there might not be a device with enough computational power to perform strategy synthesis for the entire ensemble. Thus, a decentralized approach where no component has complete control over the other components is desirable. One could call this *run-time modularization*. Yet there is another aspect to distributedness: Consider that the devices are likely to be produced by different vendors. Thus, if there is a central controlling component, its manufacturer will have a lot of power: This company can control whether, when, and how all devices, even those of other manufacturers, are used. The company could then, for example, favor its own devices in a certain situation even though the devices of another

[3]Notice that in linguistics, persistent actions are referred to as *durative*, while short actions are called *perfective*. However, we do not borrow from this terminology because in AI planning – which we will introduce in Chapter 3 – durative actions are those attributed with a fixed timespan.

vendor are better suited for the task to be carried out. This is not desirable – neither for the vendors, nor for the users. Instead, vendors should be able to develop devices' hardware and software with as little information about other vendors' devices as possible. We call this requirement, which can be considered more of a political than a technical challenge, *design-time modularization*. As a side-effect, design-time modularization improves maintainability and reusability of software that controls the devices.[4]

We now present the two key research questions that will guide us throughout this thesis:

1. Is it possible to engineer a system for ad-hoc device cooperation in smart environments in a fully distributed fashion?
2. Do users accept the assistance such a system can provide?

The first question directly follows from the set of requirements discussed above: Because we identified *distributedness* as the most important requirement, we concentrate our efforts on developing a *distributed* control system for smart ad-hoc environments. To the best of our knowledge, this is the first attempt to control devices in smart environments in a completely distributed fashion. Of course, the approach we take should fulfill the other requirements as well. The second question is also fundamental because in our approach, each component has only partial knowledge of the world. This sometimes results in suboptimal system behavior. Whether users accept such suboptimal assistance and under which circumstances they accept it has not been investigated for smart ad-hoc environments so far.

2.3 Which Kind of Assistance in Terms of Wandke's Framework?

Recall Wandke's assistance framework which we introduced in Section 1.3. Based on our survey of scenarios, we form a hypothesis which kind of assistance in terms of this framework is most appropriate for smart environments. Three dimensions must be taken into account: the *level of decision making/action selection*, *adjust-*

[4]A similar design principle is *loose coupling* in the service-oriented paradigm [Erl 2007]. However, *loose coupling* is a precisely defined term that emphasizes concepts such as *service contracts*, which we do not regard here. Hence, design-time modularization can be viewed as a more general term.

ment, and *initiative*.

Before we form our hypothesis with the help of the 15 scenarios analyzed, we take a look at Bellotti and Edwards' statement regarding which kind of assistance is preferable in our domain [Bellotti & Edwards 2001]:

> "In short, systems cannot just do things based on context awareness; rather, we argue that they are going to have to involve users in action outcomes if they are to be acceptable."

Here, Bellotti and Edwards suggest that the user should be in control to initiate system actions. In other words, regarding *initiative* they believe that *passive assistance* is preferable over *active assistance*, and regarding the *level of decision making and action selection* they argue that *delegation assistance* or a lower level is preferable to levels that give the system more autonomy, such as *informative execution assistance*. Keeping this in mind, we now form our hypothesis based on the scenarios we analyzed.

Which level of decision making and action selection? Although 3 of the 15 scenarios analyzed in Section 2.1 belong to the level *supply assistance*, we believe this level offers too little assistance in an environment equipped with lots of technical infrastructure. Indeed, one could call such a system context-sensitive because it only presents those actions that are currently executable instead of presenting all actions, but there is nothing "smart" about it. The user will still be overwhelmed by the options s/he has. *Silent execution assistance*, on the other hand, is definitely too much automation: Many studies have shown that the user must always be in control to intervene if a system does not behave according to her/his expectations [Muir 1994, Röcker *et al.* 2005]. Thus, for smart environments, the levels in between should be considered.

Recall Weiser's vision which we introduced in Section 1.1. If devices in smart environments really are to "weave themselves into the fabric of everyday life until they are indistinguishable from it", as Weiser said in [Weiser 1991], *filter assistance*, *adviser assistance*, and *delegation assistance* are not appropriate. The reason is simple: These levels force the human to pay attention to the system all the time. Such a system will not execute any actions itself, it just makes suggestions that the human can follow or decline. In any case, an action by the human is required. This is not the calm, unobtrusive technology Weiser had in mind.

Take-over assistance does not require the human to take action. The human need only intervene if s/he thinks the suggestions the system made are not correct. The problem here is that the system must always give the human time to intervene before executing an action. This will introduce a certain delay before every action which may frustrate the user. The vast majority of the scenarios analyzed in Section 2.1, 12 out of 15, belong to the level *informative execution assistance*. This leads us to the hypothesis that for smart environments, *informative execution assistance* is appropriate: The user always receives adequate feedback about the system state, which is important if the user is to accept the system [Parasuraman 1997]. Actions are executed without delay while at the same time leaving the user in control to intervene if necessary.

Which kind of adjustment? 14 of the 15 scenarios analyzed in Section 2.1 belong to the category of *fixed assistance*. This probably has the reason that fixed assistance is easier to implement than the other forms of adjustment – in fact *no* adjustment takes place. Hence, this solution is obviously driven by the preferences of the system designers. Whether it is also preferred by the *users* is a question we will try to answer in Chapter 6. For now, we form the hypothesis that *fixed assistance* is appropriate for our domain.

Which kind of initiative? 12 of the 15 scenarios analyzed in Section 2.1 belong to the category of *active assistance*. This is not surprising, since in smart ad-hoc environments the typical situation is that the user walks into a room with her/his mobile devices and immediately wants to start using the device ensemble. S/he may not know how to get the system to do what s/he wants it to do or, in some cases, may not even be aware of the assistance system. Furthermore, the user may not always have spare time or cognitive resources to devote to the assistance system. Thus, our hypothesis is that *active assistance* is preferred by users in smart ad-hoc environments.

To summarize, we hypothesize that *informative execution assistance* is the best level of assistance in smart environments, that *fixed assistance* is appropriate and that users prefer *active assistance*. Notice that this hypothesis is not in line with Bellotti and Edwards' statement introduced above. It is, however, in accordance with Weiser's vision of unobtrusive assistance. The results of the analysis in this chapter have shown how the term *unobtrusive* should be understood in the context of smart environments: *Unobtrusive* does not imply that the user should be unaware *that* s/he is being assisted. The key fact is that s/he should not have to know *how* s/he is assisted. This comprises the following criteria:

- The user should neither have to care about the technical details which are necessary to get the devices in the environment to fulfill her/his goals nor how the assistance works.
- The user should able to attend to the task s/he actually wants to perform, such as giving a talk.

Whether our hypothesis is correct will be investigated in the user study in Chapter 6.

2.4 The Device Ensemble – A Multi-Agent System

A system that conforms to our hypothesis, i.e. that provides informative execution assistance and that initiates assistance itself rather than leaving this task to the user, can be regarded as a multi-agent system. Taking this viewpoint has two benefits: First, it allows us to embed our own research efforts into a well-established conceptual framework. Second, we can build on prior work in this field. Wooldridge gives the following definition of multi-agent systems [Wooldridge 2001]:

> "Multiagent systems are systems composed of multiple interacting computing elements, known as agents. Agents are computer systems with two important capabilities. First, they are at least to some extent capable of autonomous action – of deciding *for themselves* what they need to do in order to satisfy their design objectives. Second, they are capable of interacting with other agents – not simply by exchanging data, but by engaging in analogues of the kind of social activity that we all engage in every day of our lives: cooperation, coordination, negotiation, and the like."

Devices in smart ad-hoc environments must be capable of autonomous action, too. As explained in Section 2.2, the system designer cannot precompile devices' action sequences due to the dynamic ensemble structure. Furthermore, the user may not be able or willing to tell the devices what to do. This means that the devices themselves must generate a strategy at run-time. This strategy cannot be found by one agent alone. Instead, the device ensemble as a whole must strive to fulfill the user's goals. To do this, the devices must communicate with other devices and cooperatively decide which actions to take. There are many ways to do this – we introduce the most common in Chapter 3. Thus, the devices fulfill all requirements according to Wooldridge's definition and can be regarded as *agents*. The device ensemble as a whole can be seen as a *multi-agent system*.

This discussion leads us to the technical requirements the devices have to fulfill: The ability to act autonomously and to interact with other agents requires that each device in the ensemble, even a canvas, is equipped with at least a tiny processor and some memory to be able to perform elementary computation. Furthermore, all devices must be connected to a common network. These assumptions may sound somewhat daring, but they are in line with the trend in other fields such as home automation and vehicle telematics: The idea in home automation is to connect all major appliances in the house (including tiny devices like lights and switches) via a bus system such as EIB (European Installation Bus) or its successor KNX [KNX 2009] that facilitates controlling and reprogramming them. One application in vehicle telematics is to connect cars on the road via ad-hoc networks to enable their drivers to share safety information. With the ongoing miniaturization of computing devices this will become more common in other areas, too.

2.5 Chapter Summary

In this chapter, we have analyzed 15 smart environment scenarios from the literature in order to develop a set of requirements that a suitable strategy synthesis mechanism for smart environments should meet. As a result of this analysis, we identified the requirements *spontaneity*, *action sequences*, *rationality*, *flexibility*, *robustness*, *support for persistent actions*, and *distributedness*, which we consider the most important requirement. With the help of the 15 scenarios, we have furthermore investigated which kind of assistance in terms of Wandke's framework is feasible in our domain. We have formed the hypothesis that *informative execution assistance* is a good level of decision making and action selection. Concerning adjustment, *fixed assistance* is likely to be appropriate. Concerning initiative, *active assistance* is preferable over *passive assistance*. Finally, we have argued that a system that fulfills the criteria just mentioned can be seen as a *multi-agent system*, while the devices that constitute this system can be seen as *agents*. This enables us to draw on previous work in the field of multi-agent systems. In the following chapters, we develop a fully distributed system for user assistance in ad-hoc environments based on the requirements identified in this chapter. We furthermore investigate whether users accept such a system, and under which circumstances they accept it. We start off with a review of related work in the next section.

CHAPTER 3

Related Work

Contents

This chapter presents existing control strategies from various fields in order to discuss which of them are suited for the strategy synthesis in smart environments. A well-known distinction in the field of multi-agent systems is that of reactive vs. deliberative control [Ferguson 1992, Wooldridge 2001]. It is used to categorize different approaches to agents' decision-making. Weiss gives the following definitions [Weiss 1999]:

> "Deliberative – Based on or requiring the manipulation of symbols. Usually contrasted with [...] reactive."
>
> "Reactive – (Of agent behaviour) Capable of maintaining an ongoing interaction with the environment, and responding in a timely fashion to changes that occur in it. (Of agent architectures.) An architecture that includes no symbolic representations and does no symbolic reasoning."

Deliberative control is occasionally referred to as representational, and reactive control is also called behavioral [Küngas 2002] or tropistic [Wooldridge 2001]. In Section 2.4, we have argued that device ensembles in smart environments can be regarded as multi-agent systems. We will thus use the deliberative/reactive distinction to categorize related work in this section. This allows us to classify approaches used in the domains of multi-agent systems and smart environments in a common, well-known categorization which is relevant to our field.

Deliberative and reactive approaches have different features: Deliberative approaches can find solutions to complicated problems, while reactive approaches are able to respond to unforeseen situations quickly. For many smart environment scenarios both of these features are beneficial. Fortunately, a number of approaches exist that combine the benefits of the reactive and the deliberative paradigm. They are called *hybrid*. Hence, deliberative and reactive control can be seen as two end points of a continuum, and the search for an appropriate strategy synthesis mechanism for smart ad-hoc environments can be seen as the search for the optimal point in this continuum.

In the following section, we introduce different reactive and deliberative approaches and describe state-of-the art research projects that have used them. If an

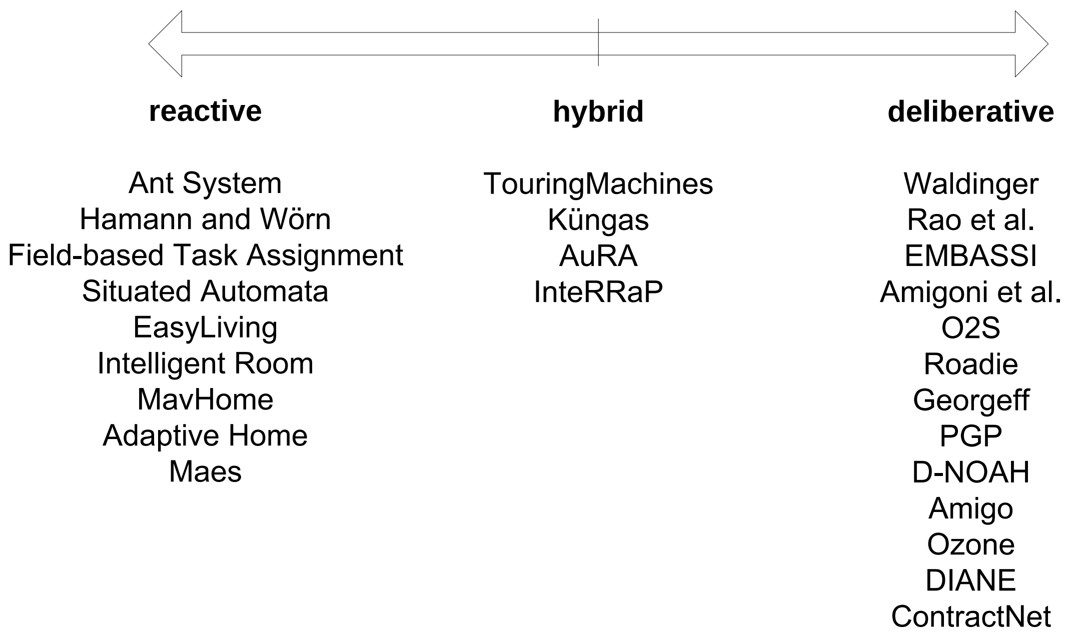


Figure 3.1: The continuum of reactive and deliberative control.

approach has been successfully used in smart environments projects, we describe those rather than applications in other domains as they are especially relevant for the work in this thesis. However, there are also some approaches that have not been used in smart environments so far but deserve a closer look. Most of them are from the field of multi-agent systems. We discuss them in this section, too. We then elaborate on the benefits and shortcomings of deliberative and reactive control. Afterwards, we introduce hybrid approaches that are able to overcome some of the shortcomings and are thus suited better for smart environments. This discussion will motivate our choice of control strategy for smart environments in Chapter 4. Figure 3.1 gives an overview of the projects discussed in this section and classifies them with respect to the continuum of deliberative and reactive control.

3.1 Deliberative Control

3.1.1 Theorem Proving

An example for deliberative control is theorem proving. In this approach, an agent's world model is represented as a knowledge base of logical formulas. Selecting an action for the agent to perform corresponds to applying a set of deduction rules to this knowledge base. If the proof succeeds, the agent can derive a formula which states that a certain action should be executed.

Waldinger [Waldinger 2001] has used theorem proving for service composition. Agents consist of services, and each agent's capabilities (i.e., the services it offers) are formulated as axioms. All axioms together constitute the application-domain theory. A query for a composite service is now formulated as a theorem, and a theorem prover proves that the theorem follows from the axioms in the application-domain theory. Certain symbols in this theory are linked to agents, and whenever one of those symbols is used in the proof, its agent becomes active. This happens for example if information that can be provided by the agent is needed during the proof. The answer to the query is then extracted from the proof.

A similar approach is that of Rao et al. [Rao *et al.* 2006]. They use Linear Logic (LL) to compose semantic web services in DAML-S (DARPA agent markup language for services) [Paolucci & Sycara 2003]. To this end, the semantic description of the existing web services (DAML-S service profile) is translated into extralogical axioms of LL. The user can now request a composite service in the form of an LL sequent that must be proven. The LL Theorem Prover then tries to answer the request by composing existing services using theorem proving. During this process, a semantic reasoner can be queried that performs subtyping inference using an ontology. If the proof can be completed successfully, a process calculus presentation is extracted from the proof, which can then be translated to a DAML-S service model or BPEL4WS (Business Process Execution Language for Web Services) [Juric 2006].

3.1.2 Planning

Another means of deliberative control is planning, also known as means-end reasoning [Wooldridge 2001]. The key elements in planning are *operators*. These operators are actions that are described in terms of *preconditions* and *effects*. Preconditions and effects are conjunctions of literals. A literal is a positive or negated atom.[1] For the action to be executed, its preconditions must hold. After the execution of the action, its effects hold. Consider the simple planning operator *Canvas-Down*:

[1]We describe these terms formally in Section 4.1.1.

```
(:action CanvasDown
  :parameters (?c - Canvas)
  :precondition (not (CanvasDown ?c))
  :effect (CanvasDown ?c))
```

This operator is described in PDDL [Ghallab *et al.* 1998], a language widely used for planning problems. It describes the action of lowering a canvas. It has a single precondition which states that the canvas must not be lowered for the action to be executed. After its execution the world state will have changed: now the canvas is lowered.

A planning problem consists of a domain description, a set of objects, a set of true literals specifying the initial world state (all literals not mentioned are assumed to be false) and a set of literals specifying the goals of the planning process. The domain description is a set of operators. Objects are used to instantiate planning operators: All variables in operator descriptions are bound to an object. The variable *?c* in the operator description of *CanvasDown* is instantiated with all objects of type *Canvas* defined in the problem description. Thus, for each *Canvas* object one instance of the *CanvasDown* operator is generated. To solve a planning problem means to find a sequence of instantiated operators (a *plan*) which transforms the initial world state into the goal state.[2]

The possible actions of devices in smart environments can be modelled as planning operators. This has the advantage that user assistance can be very flexible. Whenever a new user goal becomes apparent, a planner can consider all possible actions of all devices in the ensemble and search for a sequence that fulfills the goal. The action sequences need not be precompiled by a domain expert. Every device can carry descriptions of all its possible actions. Upon entering a new environment, it can provide these descriptions to the devices already present. This way, the device ensemble is constructed of modular pieces and can be dynamically extended. In principle, this way of modelling is not confined to centralized strategy generation like classical planning: It can also be used with a decentralized strategy synthesis mechanism. Modelling device actions as planning operators thus supports *run-time modularization*. Furthermore, a planning operator is defined in terms of its interfaces, i.e. its preconditions and effects. The actual implementation

[2]A comprehensive introduction to planning is beyond the scope of this thesis, but can be found in [Russell & Norvig 2010].

of the action is hidden. Thus, authors of planning operators do not need to know the internals of actions written by different developers. To create actions that are interoperable, they just need to make sure that the interfaces, i.e. the preconditions and effects, are compatible. Thus, modelling device actions as planning operators also supports *design-time modularization*. This way of modelling thus fulfills both parts of the requirement *distributedness* identified in Section 2.2. Unfortunately, as will become clear in the following, there is currently no planning mechanism that fully supports this requirement.

Planning as a control strategy has been successfully employed in a number of smart environments projects. One of them is the EMBASSI project [Heider & Kirste 2002] which consisted of several sub-projects. For example, the aim of the private household sub-project was to assist users in a home entertainment scenario. All devices carry declarative descriptions (precondition/effect rules) of the actions they can perform and upload them to a central controlling component. For strategy synthesis, EMBASSI introduced the concept of *goal-based interaction*, which makes controlling a device-rich environment much easier for the user. An example: In order to start a movie, the user no longer has to figure out which buttons to press on the remote control, but can state the goal *I want to see Terminator now!* declaratively. That means the user only has to know *what* s/he wants, but no longer has to care *how* to achieve this goal – a shift from the functional to the goal-based paradigm. When the user utters a goal, the controlling component tries to generate an action sequence for the devices to fulfill this goal. To this end, partial-order planning techniques are used. Once an action sequence has been found, it can be executed autonomously by the device ensemble, without user intervention.

A similar approach to the one pursued in EMBASSI is that of Amigoni et al. [Amigoni *et al.* 2005]. Unlike EMBASSI, it is not based on partial-order planning, but on a distributed version of hierarchical task networks called D-HTN (Distributed Hierarchical Task Network). But as in EMBASSI, the devices themselves provide descriptions of the actions they are able to perform. In D-HTN, these descriptions are given in the form of HTNs. These are decompositions of higher-level tasks which can be used by a central planning component in order to construct a plan for the whole ensemble. When constructing a plan, the planner can query the available devices for suitable decompositions. Should there be more than one decomposition, it can choose the most appropriate one according to values set by the system designer. For example, the task *Request* can be decomposed by a phone agent into *CreateRequestMessage*, *SearchPhoneNumber*, *MakeCall* or by a fax agent into *CreateRequestMessage*, *SearchPhoneNumber*, *SendFax*. *MakeCall* has a value of 800 and *SendFax* has a value of 200. Thus, the planner will decide to make a phone call rather than send a fax because it is more effective.

Another very similar approach is that of Saif et al. [Saif *et al.* 2003], called O2S. Again, user goals are represented explicitly. A goal can be viewed as a higher-level function which is to be decomposed into a set of lower-level actions (similar to the HTN approach). There might be several ways to fulfill a goal, and all of those candidates are represented in a goal tree. Choosing the best action sequence to fulfill a goal then corresponds to selecting a path through the goal tree. As in D-HTN, this choice is made according to values specified by the programmer. Furthermore, O2S' architecture spans across a network of interconnected devices. This way, one device can query other devices for suitable decompositions if it cannot fulfill a goal itself.

Roadie [Lieberman & Espinosa 2006] is a system that aims at assisting the user in a home entertainment scenario. All devices in the home are connected to a central component, supplying descriptions of the actions they are able to perform. As the approaches above, Roadie is based on explicit user goals. Roadie has a user interface where users can enter what they would like to do, e.g. "I want to watch a movie". Another means of input is the actual user interface of a device (e.g. buttons). This input is sent to a plan recognizer called *EventNet*, which transforms it into a goal. To this end, it matches the input sequence against a knowledge base of natural language sentences. This way it can find possible antecedent or subsequent actions that might occur, which serve as user goals. An example: If the user turns on the DVD player, the system finds several events that might follow in its knowledge base, e.g. *leave the room* or *hit play*. Matching these possibilities against the devices' capabilities yields *watch a movie on DVD*, *record a DVD movie* and *listen to a music CD*. The system now asks the user to pick one of those alternatives and the user chooses the second one. Roadie now generates a plan consisting of several steps such as *Open the DVD player door*, *Select the DVD player output that connects to the speaker* and *Insert the movie DVD* using the Graphplan planner [Blum & Furst 1995]. Some of these actions it can perform on its own, while it instructs the user on how to accomplish the rest. Notice that in Roadie the possible goals are not defined in advance but generated using a knowledge base, which makes this approach very flexible.

The problem with those approaches is that the planning process is centralized. There is one component that needs to have complete knowledge of the planning domain: the goals, the complete world state and the other components' planning operators. This conflicts with our requirement *run-time modularization*. There are approaches in other fields than smart environments that have tried to distribute planning, but all of them rely on either of two assumptions:

1. Agents all have their own goals. I.e., they have no common goals that can only be achieved by working together. Specifically, no action sequences need to be generated cooperatively as in our domain. Cooperation mainly takes place to schedule actions of different agents in order to avoid conflicts, as in [Georgeff 1988]. Another motivation for cooperation is that agents can perform tasks better if they cooperate (e.g. they can avoid redundant execution of the same action by different agents). An approach that accounts for this is Partial Global Planning (PGP) [Durfee & Lesser 1991]. In either case, agents can in principle reach their goals on their own, thus cooperation is not essential for their success.
2. Agents do work together on a common goal, yet the planning problem is hierarchically structured. In other words, there must be information available on how abstract tasks can be decomposed into more concrete subtasks, and one or more agents must have knowledge about this hierarchical structure. These agents can then distribute subtasks to other agents and collect and assemble the results the agents pass back. An example is D-NOAH which supports distributing planning control across several agents [Corkill 1979].

Both assumptions do not hold in our case: Neither do agents have their own, largely independent goals, nor is the planning problem hierarchically structured. Thus, the existing distributed planning approaches are not suited for the domain of smart ad-hoc environments.

3.1.3 Matchmaking

The matchmaking approach requires a library of abstract plans the designer has to specify. At run time, these are matched against the descriptions of services available in the environment.

The approach of the Amigo project [Vallée *et al.* 2005] is to automatically compose device services, so that users can benefit from higher level services in a smart environment. In the composition process, context information such as the user's location, current needs and preferences is used. The architecture is centered around a service infrastructure which keeps track of available devices and manages the services they offer. To fulfill a user's goal, they use a predefined abstract plan (task) description and perform a task matching between the task description and the service description model. The context information is included through a composition algorithm based on a constraint problem solver.

The Ozone project [Issarny *et al.* 2005] developed a framework which is quite similar to the approach of the Amigo project. This framework is called WSAMI (Web Services for AMbient Intelligence) and comprises a declarative language for the description of web services and a middleware that enables service composition depending on the context. For this to work, the developer of a composite service must specify abstract interfaces of atomic services the composite service must call when executed. Through the WSAMI middleware these interfaces can then be matched against the interfaces of existing services at run-time in order to instantiate the service. Interfaces match if the documents they relate to are syntactically equal. To keep processing costs low, the Ozone team even goes a step further: The documents even have to be identical, that is, have the same URI. This solution is, of course, not very flexible and not suitable for dynamic environments.

In DIANE [Küster *et al.* 2007], services are described in the service description language *DSD* (Diane Service Description). At the heart of DIANE's architecture is a central broker consisting of several agents. These agents manage and distribute services to clients in the following way: A client can ask a request agent for a service, which in turn calls other agents to search for available services, chooses a suitable service and invokes it. Services can be either atomic or composed of several atomic services. For the latter case DIANE pursues an approach that integrates service composition, discovery and matchmaking. Service requests are described via the effects they should fulfill. Then a suitable composite service is built in three steps:

1. All available service offers that fulfill some of the effects are picked. Variables are not yet instantiated.
2. All possible compositions of these offers are computed. The ranges of the variables are lowered by computing the cuts on the parameters if services depend on one another.
3. The variables are filled in such a way that the service composition yields the best possible results.

3.1.4 ContractNet

The ContractNet protocol [Smith 1980] specifies how a group of agents can work together to solve a task they could not solve alone. Upon reception of a new task, an agent generates a task announcement and sends it to other agents. If one of those agents can help to solve the task, it sends back a bid stating which capabilities it has to solve the task. The agent that sent the announcement may receive several bids and will then choose the most suitable one. The "winning agent" is then contracted, i.e. chosen to fulfill the task. It can now either solve the task alone or split it into subtasks and contract other agents. The problem with this protocol is that it can not easily be distributed. Suppose a task is formulated as the goal of a planning problem and the agents' capabilities are formulated as planning operators using preconditions and effects. In order to find good results, hierarchical knowledge must be available. I.e., operators must be decomposable into finer-grained operators, and those decompositions must be known to an agent so it can award the subtasks to other agents. However, in very dynamic environments, such decompositions will often not be available. Usually, agents will only know about their own capabilities, not the capabilities of other agents. If no hierarchical knowledge is available (i.e., all tasks are elementary and cannot be split into subtasks), the planning horizon is 1, which is very small. An agent which can perform an action with an effect that corresponds to a goal must award tasks that can fulfill its preconditions to other agents. These in turn will award tasks that can fulfill their preconditions to other agents and so on. In this case, ContractNet is a purely local mechanism as every agent considers only its direct predecessors. Thus, the resulting action sequence will likely be very suboptimal.

3.1.5 Benefits and Shortcomings of Deliberative Control

As stated in the definition above, deliberative approaches are based on the manipulation of symbols. The human brain also manipulates symbols. Thus, deliberative approaches resemble processes which take place in the human mind when we think or plan what to do [Wooldridge 2001]. Like the human brain, they can encode complex information in symbols and perform complex manipulations on those symbols, which enables them to solve sophisticated problems.

Yet they also have a number of shortcomings. The first is computational complexity. Deliberation can take a long time (in fact, it may not even terminate), and the world may change in the meantime, rendering the plan just generated useless. In [Maes 1990b], Maes remarks the following:

> "Although the deliberative thinking approach has proven successful for certain other tasks, only poor results have been obtained with planning, in particular, when applied in real autonomous agents operating in complex, dynamic environments. The few systems built show major deficiencies such as brittleness, inflexibility, and slow response times. They also spawned a number of theoretical problems such as the frame problem and the problem of non-monotonic reasoning which so far remain unsolved in satisfactory ways [...]. More recently, some researchers have been viewing this as evidence that it is unrealistic to hope that action-oriented tasks can be successfully implemented by a deliberative machine in real-time."

Furthermore, due to their computational complexity, deliberative approaches do not scale well. Wooldridge pointed out another problem: For many environments, it is not clear how the environment should be mapped to the agent's internal world model [Wooldridge 2001]. Thus, it is hard for the system designer to decide what should be modelled and how. An optimal modelling would imply that the developer can foresee every problem that might occur. This is, of course, impossible. Moreover, the component that does the deliberation, be it a theorem prover, a planner, or a matchmaker, must have a lot of knowledge about the domain, including the capabilities of other agents. In fact, complete domain knowledge is preferable. This makes it hard to distribute deliberative approaches.

What are the implications of this discussion for smart environments? Let us reconsider the requirements introduced in Section 2.2: *Action sequences* need to be generated, but these are rather short (about seven actions). In other words, deliberative control mechanisms can solve more complex problems than required for smart environments. On the other hand, in smart environments, *robust* systems are required. Yet deliberative approaches typically suffer from brittleness. The most important requirement introduced in Section 2.2 is *distributedness*, and distributing deliberative approaches is not easy. This leads us to the assumption that deliberative approaches are not the optimal control strategy for smart environments. Thus, it is beneficial to look at reactive approaches.

3.2 Reactive Control

3.2.1 Swarm Intelligence

The concept of swarm intelligence was introduced by Beni and Wang [Beni & Wang 1989]. According to them, intelligent swarms are groups of similar or equal robots each behaving in such a way that intelligent behavior emerges as a result of the local interactions of the group's members with one another and with the environment. Important features are decentral control and asynchronicity. This concept resembles the behavior of flocks of birds, schools of fish, insect swarms and ant colonies. It has various applications, e.g. in optimization. An example is the Ant System, an algorithm which computes near-optimal solutions to the Traveling Salesman problem [Dorigo *et al.* 1996].

3.2.2 Embodied Computation

A concept very similar to swarm intelligence is embodied computation, a term coined by Hamann and Wörn [Hamann & Wörn 2007]. The main difference is that swarm intelligence is only interested in the swarm as a whole which constitutes one huge computing device. In contrast, embodied computation regards the swarm on the microscopic level (the individual robots) as well as on the macroscopic level (the swarm as a whole). Embodied computation has been used to approximate a solution to the Steiner Tree problem: A given set of points is to be connected by lines of minimal length. This is very similar to the Minimum Spanning Tree problem. The difference is that it is allowed to add extra points.

3.2.3 Field-based Task Assignment

Field-based task assignment (FiTA) by Weyns et al. is another similar approach [Weyns *et al.* 2008]. It has been applied in a transportation scenario: Several AGVs (automatic guided vehicles) capable of picking up, carrying and dropping a load, are distributed in a warehouse. In addition, several loads are distributed which must be carried to certain drop locations by the AGVs. The assignment of an AGV to a load is carried out using a field-based approach: AGVs and loads emit fields. An AGV is attracted by a load and rejected by another AGV. AGVs combine the fields they perceive and follow the gradient of the combined fields. This strategy leads to loads being picked up while at the same time preventing several AGVs from driving to the same load.

Swarm Intelligence, Embodied Computation and FiTA all exhibit the following features:

- Failure of a single or a few agents does not cause the system to break down. It is thus very robust.
- As agents' interactions are purely local, the solutions are suboptimal.
- Because these approaches are local, the algorithms scale very well.
- The physical world is an important part of the problem to be solved. The positions of the agents in the environment can be viewed as approximate solutions to the overall problem. On the other hand, this means that these approaches can only solve geometrical problems, i.e. problems with an inherent spatial layout.
- Agents are not specialized to certain tasks, each task can be solved by any agent. Thus, the approaches cannot solve complex tasks that require distribution among several agents.

3.2.4 Condition-Action Rules

Condition-action rules are a very simple way of controlling agents and smart environments. This approach is based on a set of rules each consisting of a condition and an action: Whenever the condition holds in the world model, the rule fires and the action is executed. These rules can either be fixed, i.e. specified by the system designer, or learned by observing the user. Fixed rules have the advantage of being very simple and intuitive, yet this approach also very inflexible: Whenever the system is extended by new devices or services, new rules have to be added manually by the developer. Learning the rules from the user is more flexible, but it takes a significant amount of time before the system functions properly because it has to learn. Another drawback of the rule approach is that it is fully centralized.

Condition-action rules are essentially the core of the situated automata paradigm introduced by Rosenschein and Kaelbling [Kaelbling 1991]. An agent is seen as a function mapping a stream of inputs from the environment to a stream of actions. The agent is modelled as a finite state machine which is expressed as a fixed sequential circuit. This circuit can be decomposed into two components: a perception and an action component. The perception component maps an input and the internal state of the agent to a new internal state. The action component maps the input and the old state of the agent to an action output. The rules that make up these two components can be specified in two high-level languages – Ruler for the perception component and Gapps for the action component. Those rules are then compiled into a sequential circuit at design time. Thus, the rules are not represented explicitly in the agent, but implicitly in the agent's circuitry.

The EasyLiving project [Brumitt *et al.* 2000] used fixed behavior rules that cause things to happen automatically in a smart home when their condition is fulfilled. For example, when a user moves from the PC to the couch, the content of the PC screen will automatically be transferred to a big wall screen because two conditions are fulfilled: The wall screen is available and it is in the user's field of view.

3.2.5 Subsumption Architecture

The subsumption architecture [Brooks 1990] was developed by Brooks for robot control. It is based on the assumption that robot control should be organized in a hierarchy of different behaviors such as "avoid objects", "wander" and "explore". All layers in this hierarchy react to stimuli and produce output (i.e. actions the robot should perform) simultaneously. Each layer can be seen as an agent that acts autonomously. Lower layers represent more elementary behaviors and can suppress the output of higher layers which represent more abstract behaviors. This ensures that the robot can perform complex behavior while still being able to react to unforeseen situations very quickly. For example, the output of the "avoid objects" layer will suppress the output of any higher layer such as "explore" because the most important action of a robot is to avoid obstacles. Brooks implemented each layer as a finite-state machine. Thus, the control architecture is purely reactive and works without any symbolic reasoning.

The Intelligent Room project [Coen 1997] used an architecture to control a smart room that was inspired by the subsumption architecture. This architecture consists of a number of agents controlling the devices in the room. The agents are organized in a layered architecture with different levels of abstraction. The agents in the lowest layer (which is called the *Scatterbrain*) are called *SodaBot* agents. They control and interconnect the devices in the environment. Agents in higher layers can use a combination of agents in a lower layer in order to perform more sophisticated tasks. These combinations are hard-wired by the system designer. For example, the SodaBot Netscape agent communicates with the SodaBot Display agent to make sure that web pages are displayed in an area in the room that is visible for the users. These two agents are in the lowest layer. Any agents on subsequent layers that use the Netscape agent need not worry about information being visible to the users as the lower-level agents deal with this task autonomously. The agents on the highest level are invoked by the user, for example via speech.

3.2.6 Pattern Matching

Patterns observed in past interactions of user activities in a smart environment can be used to automate the user's interactions with the devices in the environment. User activities as well as the user's device interactions are constantly recorded. This is called the *history*. If a sequence of activities is detected that equals a sequence in the history (i.e. it has been observed once or several times before), this is called a *match*. The history may contain several device interactions that followed the match in the past. The device interaction that directly followed the match most frequently is the most likely to occur at this point. Thus, it can be automated.

Pattern matching has been used in the MavHome project [Das *et al.* 2002], the smart home project related to the scenario in Section 2.1.5. However, the MavHome researchers refrained from automating the device interactions most likely to occur because a wrong prediction can be very annoying for a user. Instead, the input stream is searched for significant episodes. A significant episode is an ordered, partially ordered or unordered set of device events that reoccurs periodically, such as the episode *CoffeeMakerOff*, *KitchenLightOff*, *KitchenScreenOff* occurring daily. Significant episodes are extracted from the input stream of events using the minimum description length (MDL) principle: Significant episodes are patterns that minimize the description length of the history if each occurrence of the pattern is replaced with a pointer to this pattern. Thus, the frequency of occurrence in the history and the pattern length are the two important parameters when determining significant episodes. They are then automated as this significantly reduces the number of device interactions for the user.

3.2.7 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models that resemble natural (biological) neural networks in their architecture and function. They consist of a number of interconnected computational units called *neurons* and can be used to approximate complex, usually unknown functions. ANNs are organized in layers – an input layer, an output layer, and possibly a number of hidden layers. The strength of the connections between the neurons and the thresholds for a neuron to become active may vary – and this property is used for training the ANN, i.e. approximating the target function. In *supervised learning*, the ANN is trained using a set of training data consisting of an input and an expected output. The inputs are fed into the ANN. The weights and thresholds are then adjusted in order to minimize the ANN's prediction error – the discrepancy between the expected output and the network's actual output.

An ANN has been used to automate the inhabitants' device interactions in the Adaptive Home [Mozer 2005], the smart home related to the scenario in Section 2.1.6. It has been employed for lighting control. The house was partitioned into several zones and the aim was to optimize lighting and heating conditions in each zone. Two conflicting objectives had to be taken into account: maximizing the inhabitants' comfort level and minimizing energy consumption. The ANN was trained to predict the inhabitants' transitions from one zone to another in order to anticipate which zones would be occupied in the next few seconds. A device controller could then draw on this information to adjust the lights and heating through-

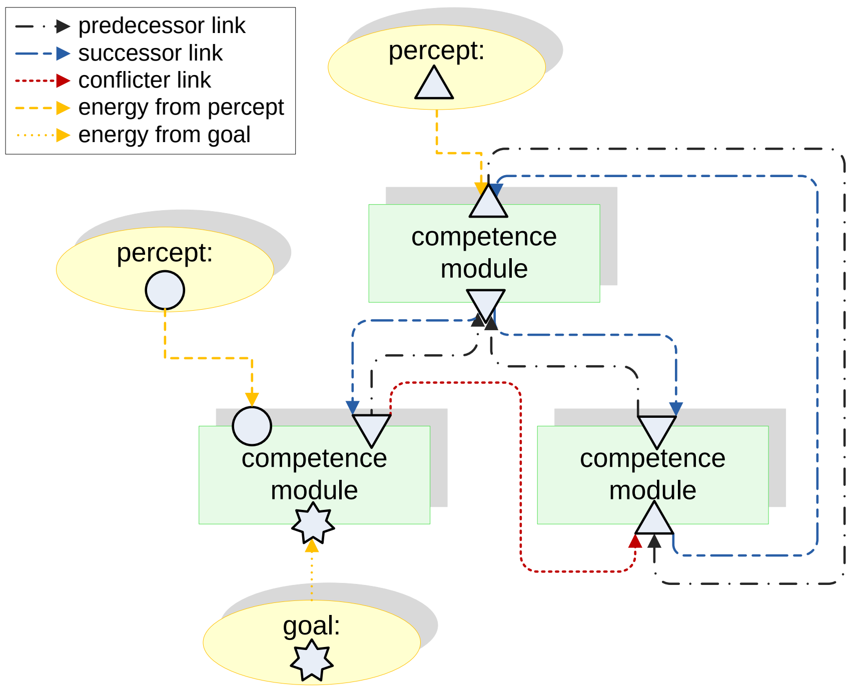


Figure 3.2: A Maes spreading activation network. The light blue shapes at the top of the competence modules represent preconditions, those at the bottom represent effects.

out the house. In order to determine which actuators should be switched on when certain zones are occupied, a reinforcement learning algorithm was used. Whenever the inhabitants showed discomfort with the decision of the device controller (e.g. manually switched on a light the device controller had not switched on), the learning algorithm would incorporate this decision into the device controller.

3.2.8 Maes' Spreading Activation Networks

Maes developed an action selection mechanism based on a spreading activation network for robot control [Maes 1990b]. Each action is described declaratively in terms of preconditions and effects, like operators in planning. These action descriptions are called *competence modules*. These competence modules are connected into a network by virtual links. There are three types of links. All of them connect equal or opposite literals and are depicted in Figure 3.2.

If a precondition of one competence module equals the effect of another competence module, a *predecessor link* connects the effect to the precondition. A *suc-*

cessor link connects the same literals, but in the opposite direction. The third type of link, the *conflicter link*, connects a precondition to an opposite effect. The modules containing the respective literals are called *predecessors*, *successors*, and *conflicters* of a competence module. The semantics are the following: A predecessor of a competence module can fulfill one of its preconditions. It can thus "help" the module to become active. A successor of a competence module can benefit from the module's execution because it "uses" one of the literals the module makes true when it becomes active. A conflicter is an opponent of a competence module as it can "destroy" a literal the module needs for becoming active.

Once the modules have been linked to one another, the network can be used to generate a sequence of actions that fulfills a goal. This is done via an energy distribution mechanism (see Figure 3.2). Here, *energy* is a numerical value that indicates how likely it is that a given competence module can help to achieve a goal given the current world state. In the following, we call a literal that is part of the current world state a *percept*. In the beginning, competence modules that have a precondition corresponding to a percept receive a certain amount of energy because these modules may be executed right away. Thus, if executed, such a module would be the beginning of an action sequence. Then modules that have an effect corresponding to a goal receive energy because they may fulfill the goal. Thus, if executed, they would be the end of an action sequence. Afterwards the modules distribute energy to other modules along the links: A module sends a certain fraction of its own energy level to a predecessor. A different fraction is sent to a successor. Furthermore, a certain negative fraction is sent to a conflicter – a conflicter loses energy as it may "harm" the module. Afterwards, each module sums up the energy it has received to calculate its new energy level. If this level is above a certain threshold (the *activation threshold*), all its preconditions are fulfilled and no other module has a higher energy level, the module is executed. Its effects become true and another selection round starts. If no module can be executed, the activation threshold is lowered for the next round. The process stops when all goals have been fulfilled.

Maes' action selection algorithm has been used for a number of purposes. Dorer [Dorer 1999] described a version of Maes' algorithm that uses real-valued instead of binary literals and used it for controlling simulated RoboCup soccer agents. In smart environments, however, real-valued literals would lead to state explosion. Singleton [Singleton 2002] used a genetic algorithm to tune the parameters of

Maes' algorithm, Decugis and Ferber [Decugis & Ferber 1998] proposed a system that learns the links between the components of Maes' algorithm at run-time. Both extensions cannot be used in ad-hoc environments as they require a training phase.

3.2.9 Benefits and Shortcomings of Reactive Control

Wooldridge [Wooldridge 2001] identifies the following advantages of reactive approaches:

- simplicity
- economy
- computational tractability
- robustness against failure
- elegance

However, they also have major shortcomings:

- Because reactive agents have no internal models of the environment, the environment itself must contain enough information to select suitable actions for the agent.
- Reactive agents base their decision-making on local information. It is difficult to incorporate non-local information. Hence, the planning horizon is necessarily narrow.
- It is difficult to build reactive agents that learn from their experience.
- A big advantage of reactive agents is that their behavior emerges from the interactions of the components' behaviors with one another and with the environment. Yet *emerges* also means that the relationship between those components cannot be fully understood. Thus, it is hard to engineer agents for specific tasks. This is necessarily a process of trial and error.
- Agents with a small number of behaviors can be built easily. Yet building agents with many behaviors is much more difficult because the dynamics of the interactions between a high number of behaviors are usually too complex to understand.

What are the implications for smart environments? In this domain, *robustness* is very important. This requirement can be fulfilled by reactive systems. However, as Hamann and Wörn have remarked, the quality of a system's solutions is lower the more robust this system is [Hamann & Wörn 2007]. This means that reactive systems likely produce suboptimal solutions. They are thus not able to completely fulfill the requirement *rationality*. An advantage, however, is that they do not perform complex computation, which is beneficial in smart environments where devices are often resource-poor. The challenge is, then, to increase the solution quality of reactive systems without sacrificing too much of their robustness. Moreover, the narrow planning horizon of the approaches introduced in this section is problematic. In smart environments, *action sequences* consisting of several actions have to be generated. This cannot be accomplished by an approach that puts all information into the environment and that relies on purely local information. Some non-local information is needed. For this, it is beneficial to have a (however rudimentary) world model. Another problem is that reactive systems with a high number of behaviors are too complex to be understood. In smart environments, each device in the ensemble has one or more behaviors (the actions it can execute). Thus, if the ensemble contains more than a few devices, there can be a large number of such behaviors. Yet usually only a fraction of them is required to form an action sequence to fulfill an open goal. Thus, it would be beneficial if we could rule out those that are not relevant in the current context, i.e. that cannot be part of an action sequence which is able to fulfill the goal.

As both deliberative and reactive control have severe shortcomings with respect to smart environments, the optimal control strategy for this domain is probably somewhere in the middle of the continuum. This is where hybrid approaches are situated.

3.3 Hybrid Approaches

To combine the advantages of deliberative and reactive approaches, layered architectures were introduced. These are capable of reactive behavior for immediate, reflex-like response to certain stimuli as well as deliberative control for achieving more complex goals. According to Müller et al. [Müller *et al.* 1994], two types are common: vertically and horizontally layered architectures (see Figure 3.3). Both types consist of several control layers, and each layer is responsible for producing a certain kind of behavior. In its simplest form, a layered architecture consists of two layers: one for deliberative and one for reactive behavior. Some scholars

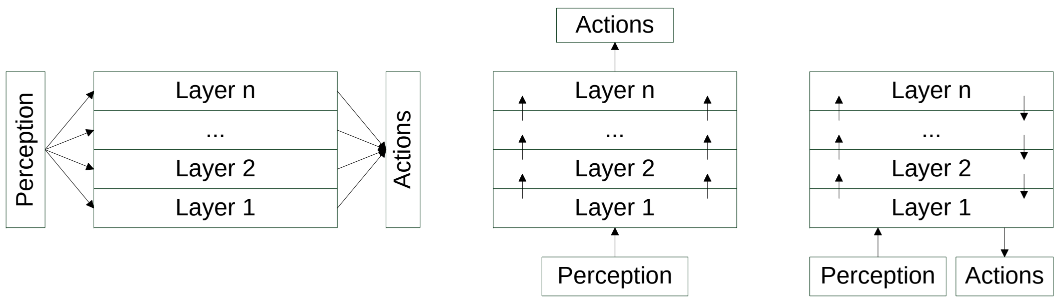


Figure 3.3: Layered architectures (cf. [Müller *et al.* 1994]): Horizontally layered (left), vertically layered with one-way control (middle) and vertically layered with two-way control (right).

even view Brooks' subsumption architecture as a hybrid approach. However, in Brooks' implementation, each layer is a finite-state machine. Thus, all behaviors are hard-wired; there is no deliberative layer.

3.3.1 Horizontally Layered Architectures

In horizontally layered architectures each layer is coupled to the perception and action components (see Figure 3.3, left). Usually all layers work concurrently and produce output, i.e. candidate instructions for the action components. Therefore, there must be a mechanism that decides which layer's output is actually sent to the actuators. This may be a controlling component or simple rules. Müller et al. [Müller *et al.* 1994] identify this as the major drawback of horizontally layered architectures: Each layer may interact with each other layer, thus if we only consider bilateral interactions, the controlling mechanism must potentially arbitrate among $n*(n-1)/2$ interactions, which makes the system very complex and hard to control.

An example of a horizontally layered architecture are Ferguson's TouringMachines [Ferguson 1992]. A TouringMachine consists of three layers: The reactive layer R provides a fast response to events the higher layers have not been programmed to deal with. The planning layer P generates and executes hierarchical partial plans. The modelling layer M models the agent and its environment. This knowledge base can then be used to predict future behaviors. All three layers have access to the perception as well as the action subsystem and concurrently produce outputs. As the outputs from different layers may conflict with each other, a control framework arbitrates between them. This control framework consists of domain-specific condition-action rules the system designer has to specify. There are two types of control rules: *Censor rules* prevent certain layers from receiving input another layer is suited better for. Likewise, *suppressor rules* suppress the output of certain layers because another layer is capable of dealing with the situation. Suppressor rules must be written by the system developer in such a fashion that they do not interfere with each other (i.e., at most one suppressor rule fires in a given situation) and that only one instruction per time slice is sent to the actuators. Thus, writing control rules is a very complex task for the developer.

Küngas also presents an architecture for robot control that consists of a behavioral and a representational subsystem which are horizontally layered [Küngas 2002]. The behavioral subsystem consists of a number of reactive entities called behaviors. The representational subsystem is based on planning. Unfortunately, the author gives no detailed explanation how the system arbitrates between the two subsystems. It seems that a component called the *Action Executor* performs this arbitration. What kind of control mechanisms the action executor employs is not explained. The paper only states that the default for the action executor is to execute plans coming from the representational subsystem. Plan execution can be interrupted by emergent behavior, presumably originating from the behavioral subsystem.

3.3.2 Vertically Layered Architectures

In vertically layered architectures, only one layer communicates with the perception components, and only one layer communicates with the action components. There are two variations: In the first, the lowest layer gets the perceptions. Control is then propagated up the layer hierarchy. The highest layer eventually sends instructions to the actuators. This is depicted in the middle of Figure 3.3. The second possibility is to couple both perception and action to the lowest layer. Upon reception of a percept, the lowest layer propagates control to the next layer etc. Decisions of the higher layers are then sent back down the hierarchy to the lowest layer, which sends instructions to the actuators, as shown in the right of Figure 3.3. In a vertically layered architecture with n layers, there are $n - 1$ interfaces between the layers. Thus, it is much less complex than a horizontally layered architecture. The drawback is that the lowest layer must be very carefully designed because anything the agent perceives or does must pass the lowest layer. Thus, it is critical that this layer does not fail.

The Autonomous Robot Architecture (AuRA) [Arkin & Mackenzie 1994] can be seen as a variation of the vertically layered architecture in the middle of Figure 3.3. It consists of three layers. Control flows from the highest layer to the lowest. The three layers are: the mission planner, the navigator and the pilot. The mission planner is concerned with very high-level mission planning. The navigator takes the specification generated by the mission planner and uses a map of the robot's environment to choose a point-to-point route that satisfies the specification. This route is then fed into the pilot segment-wise. For every segment, the pilot selects motor schemas such as *move-to-goal* or *avoid-static-obstacle* and parametrizes them so they function in the actual environment. The pilot can react to unforeseen events. It sends the respective commands to the actuators and monitors their progress in reaching the desired goal. If a failure occurs, the pilot replans, taking into account the changed conditions. Upon success the next segment of the route can be executed.

The InteRRaP agent architecture [Müller 1996] consists of three vertical layers: the behavior-based layer (BBL), the local planning layer (LPL), and the cooperative planning layer (CPL). These work concurrently and are arranged in a hierarchy – the BBL takes care of reactive behavior, the LPL performs single-agent planning, and the CPL is able to generate joint plans together with other agents. Only the lowest layer, the BBL, communicates with the sensors and actuators. Thus, the InteRRaP architecture is of the type depicted in Figure 3.3 on the right. The control flow in InteRRaP is determined by two mechanisms: If a layer is not competent to deal with a situation, it sends an *activation request* to the layer directly above it in the hierarchy. If a higher layer has made a decision, it sends a *commitment posting* to the layer below it. This layer in turn processes these commitments into lower-level plans. The commitments made by lowest layer, the BBL, are instructions sent to the actuators. Thus, activation requests flow from the bottom to the top of the hierarchy, while commitments flow back down. Each layer is associated a knowledge base which contains a world model tailored to that layer: The BBL's knowledge base contains a very primitive world model, i.e. data coming directly from the sensors. The LPL's knowledge base contains a mental model of the agent's capabilities, while the CPL contains a social model, i.e. a model of other agents's capabilities.

3.4 Chapter Summary

In this chapter, we have introduced deliberative and reactive control mechanisms and have shown that both paradigms have severe shortcomings which make them unsuitable as control strategies for smart ad-hoc environments. Shortcomings of deliberative control include brittleness, inflexibility, slow response times, and computational complexity. Furthermore, they are based on sophisticated models of the environment, which require in-depth knowledge of the domain. This makes it hard to distribute such approaches. Reactive systems, on the other hand, require an environment that contains enough information for the agent to select actions and suffer from a narrow planning horizon. Furthermore, they are difficult to fully understand because behavior emerges from the interactions of their components. This also makes engineering agents with many different behaviors difficult.

Subsequently, we have shown that by combining deliberative and reactive principles, some of the shortcomings of both paradigms can be overcome. The interesting question that remains to be answered is which hybrid approach is best suited for our domain. Hybrid approaches have traditionally been used for robot control. In this domain, enabling different behaviors is extremely important: A quick, reactive response is crucial for an agent to be able to react to unforeseen situations, e.g. to avoid an obstacle that appears unexpectedly in front of a robot. A deliberative layer, on the other hand, is important to decide what the robot is going to do in the future, e.g. for path planning.

In smart environments, we have different requirements. Obstacles that suddenly appear in front of a notebook or a projector and must be avoided are very rare. And we do not have to plan paths. In other words, there are no two fundamentally different situations, one of which requires a fast response and the other longer-term planning. If we look at the scenarios analyzed in Section 2.1, it becomes clear that we have relatively uniform planning problems that require to compose action sequences of moderate length (about 7 actions).

The main benefits of the reactive paradigm in smart environments are that it is able to find a solution without complex computations and that it is architecturally simple, which allows to assemble dynamic ensembles from a number of small components at run-time. However, a high number of behaviors may result from such an architectural design, which may deter a reactive system from finding a solution.

The challenge is thus to prune irrelevant actions in advance to restrict the search space for the reactive system. This could be achieved by a deliberative component. However, horizontally and vertically layered architectures are not suited for this because their layers are arranged in parallel, not sequentially. Thus, in Chapter 4, we introduce a different kind of layered architecture which is better suited for smart ad-hoc environments: the temporally layered architecture.

Chapter 4

The AdDCo Algorithm

Contents

In this chapter, we describe a strategy synthesis mechanism which is suited for smart ad-hoc environments based on the requirements identified in Chapter 2 and on the discussion of related work in Chapter 3.

As stated in Section 3.1.2, modelling the actions of devices in smart environments as planning operators is beneficial because it supports both parts of the requirement *distributedness* identified in Section 2.2:

- *Run-time modularization*: Each device can provide descriptions of the actions it can perform. This way, the domain description can be adapted as the ensemble itself is adapted: When a new device joins the ensemble, new operators are added to the domain. When a device leaves, the operators corresponding to its actions are deleted from the domain. In principle, the control strategy utilizing this kind of modeling can be implemented in a distributed fashion.
- *Design-time modularization*: Authors of planning operators need no knowledge about the actual implementations behind other planning operators. It is sufficient to ensure that the interfaces (i.e. the preconditions and effects of the planning operators) are compatible. Thus, planning operators written by different authors can be used together.

One strategy synthesis mechanism that allows modelling device actions as planning operators is that of Pattie Maes introduced in Section 3.2.8. It is an action selection mechanism: Provided that the *current world state* and the *user goals* are known, it selects and executes an action which leads to a new world state. Based on this new world state, another action is selected and executed. This process is repeated until the goals have been fulfilled. It is important to note that Maes' algorithm enables *goal-based interaction*, which we introduced in Section 3.1.2. This is a favorable paradigm for smart environments because the user only has to know *what* s/he wants, but not *how* to achieve it.

However, Maes' algorithm has some drawbacks: First, it is a *reactive* approach. In Chapter 3, we have argued that for smart ad-hoc environments, a hybrid approach is better suited than a purely reactive or a purely deliberative approach. Second, it is *centralized*. In its original form, it thus does not satisfy the requirement *distributedness*. Third, actions need to be *"hand-wired"* at design time, hence the algorithm does not fulfill the requirement *flexibility*.[1]

In order to adapt Maes' algorithm for the domain of smart ad-hoc environments, two things must be done: First, we must combine it with a deliberative step. This will move it from the reactive endpoint of the control continuum towards the middle where the hybrid approaches are situated. Second, we must distribute the algorithm, which also makes it more flexible. These two adaptations are described in this chapter. We start with the preliminaries needed to describe the algorithms in this chapter. We then present Maes' original algorithm before giving a detailed description of the necessary adaptations for smart ad-hoc environments. The resulting algorithm is called the *AdDCo algorithm*. Furthermore, we evaluate this algorithm against Maes' algorithm with the help of four smart environment scenarios. We then classify the AdDCo algorithm with respect to two existing agent models: On the single-agent level, it constitutes a new kind of hybrid architecture: a temporally layered architecture. On the multi-agent level, it can be viewed in terms of Wooldridge's and Jennings' four-stage Cooperative Distributed Problem Solving (CDPS) model [Wooldridge 2001]. Details follow in Section 4.6.2. Finally, we discuss how the AdDCo algorithm is able to fulfill the requirements identified in Section 2.2.

[1]What this means in detail will be explained later.

4.1 Preliminaries: Operators – Syntax and Meaning

We describe the algorithms in this chapter in a style similar to the descriptions common in the planning community. The syntax we use is based on Z [Spivey 1992], a notation commonly used for formally specifying software systems and modules.

4.1.1 Syntax

The basic building blocks of our descriptions are the following four sets:

- constant symbols, $c \in \mathit{Const}$, e.g. $[\![$`Projector1`$]\!]$, $[\![$`Canvas3`$]\!]$
- variable symbols, $v \in \mathit{Var}$, written as e.g. $[\![$`?p`$]\!]$, $[\![$`?c`$]\!]$
- predicate symbols, $p \in \mathit{Predicate}$, for instance $[\![$`CanvasUp`$]\!]$, $[\![$`DocShown`$]\!]$ (where we exclude the reserved names $[\![$`not`$]\!]$, $[\![$`and`$]\!]$, $[\![$`forall`$]\!]$)

Basically, we wish to represent the fact that preconditions and effects in operator schemes (which are introduced in Section 4.4.5) are conjunctions of function-free first-order literals. Both positive and negative literals are supported in effects *and* preconditions. Later, we will extend effects to also contain universally quantified literals. Function-free literals are literals that contain only function-free terms; terms denote objects in the universe of discourse, thus we only allow constants and variables as terms.[2]

$$\mathit{Term} ::= c \mid v, \qquad \text{where } c \in \mathit{Const}, v \in \mathit{Var} \tag{4.1}$$

$$\mathit{Atom} ::= [\![\texttt{(}p\ \ t_1 \ldots t_n\texttt{)}]\!] \qquad \text{where } t_i \in \mathit{Term}, p \in \mathit{Predicate} \tag{4.2}$$

$$\mathit{Literal} ::= a \mid [\![\texttt{(not}\ \ a\texttt{)}]\!] \qquad \text{where } a \in \mathit{Atom} \tag{4.3}$$

A positive literal is just an atom, a negative literal is an atom preceded by the negation sign.

$$\mathit{Formula} ::= [\![\texttt{(and}\ \ l_1 \ldots l_n\texttt{)}]\!] \qquad \text{where } l_i \in \mathit{Literal} \tag{4.4}$$

[2]This renders the Herbrand universe [Flach 1994] finite (as long as only finite sets of finite terms are allowed).

We will use *Formula* to represent conjunctions; an individual l_i is called a conjunct.

$$\mathit{Expression} ::= \mathit{Term} \mid \mathit{Atom} \mid \mathit{Literal} \mid \mathit{Formula} \tag{4.5}$$

When discussing aspects of this first order language, such as semantics, we will use notational conventions of [Stoy 1977], where $[\![\cdot]\!]$ is used to denote the abstract syntax tree of some object language string, see Appendix 9.1 for the details.

Par abus de langage, we write $a \in f$ or $[\![\texttt{(not } a\texttt{)}]\!] \in f$ or $l \in f$ to denote the idea that a certain positive, negative, or arbitrary literal is one of the conjuncts in a formula $f \equiv [\![\texttt{(and } l_1 \ldots l_n\texttt{)}]\!]$. We write $f \restriction l$ to represent the formula that we get by removing the literal l from f, so that for instance $[\![\texttt{(and } a\ b\ \texttt{(not } c\texttt{))}]\!] \restriction b = [\![\texttt{(and } a\ \texttt{(not } b\texttt{))}]\!]$. Also, we write $f \frown l$ to denote the extension of a formula by another literal, so that $[\![\texttt{(and } a\ b\texttt{)}]\!] \frown c = [\![\texttt{(and } a\ b\ c\texttt{)}]\!]$. Likewise, $a \frown [\![\texttt{(and } b\ c\texttt{)}]\!] = [\![\texttt{(and } a\ b\ c\texttt{)}]\!]$, and $[\![\texttt{(and } a\ b\texttt{)}]\!] \frown [\![\texttt{(and } c\ d\texttt{)}]\!] = [\![\texttt{(and } a\ b\ c\ d\texttt{)}]\!]$ Finally, we write $[\![\texttt{()}]\!]$ ($\equiv [\![\texttt{(and)}]\!]$) to denote the concept of an empty formula (a formula without conjuncts).

4.1.2 Operations on Expressions

The function $\mathit{vars} : \mathit{Expression} \longrightarrow \mathbb{P}\,\mathit{Var}$ gives the set of variables used in an expression. It is easily given by:

$$\mathit{vars}\ c = \emptyset \tag{4.6}$$

$$\mathit{vars}\ v = \{v\} \tag{4.7}$$

$$\mathit{vars}[\![\texttt{(}p\ \ t_1 \ldots t_n\texttt{)}]\!] = \bigcup_{i=1}^{n} \mathit{vars}\ t_i \tag{4.8}$$

$$\mathit{vars}[\![\texttt{(not } a\texttt{)}]\!] = \mathit{vars}\ a \tag{4.9}$$

$$\mathit{vars}[\![\texttt{(and } l_1 \ldots l_n\texttt{)}]\!] = \bigcup_{i=1}^{n} \mathit{vars}\ l_i \tag{4.10}$$

An expression is a *ground expression* if it contains no variables.

$$\mathit{GroundExpression} = \{\, e : \mathit{Expression} \mid \mathit{vars}\ e = \emptyset \,\} \tag{4.11}$$

The corresponding definitions for ground terms, atoms, literals, and formulas follow immediately.

A *binding* $\beta : Binding$, where $Binding = Var \longrightarrow Const$, is a (finite) mapping from variable names to constants. The function $subst : Binding \longrightarrow Expression \longrightarrow Expression$ describes the substitution of variables in an expression using a given binding:

$$subst_\beta\ v = \begin{cases} \beta\ v & \text{if } v \in \operatorname{dom} \beta \\ v & \text{otherwise} \end{cases} \tag{4.12}$$

$$subst_\beta\ c = c \tag{4.13}$$

$$subst_\beta [\![\texttt{(}p\ \ t_1 \dots t_n\texttt{)}]\!] = [\![\texttt{(}p\ \ (subst_\beta t_1)\ \dots\ (subst_\beta t_n)\texttt{)}]\!] \tag{4.14}$$

$$subst_\beta [\![\texttt{(not }a\texttt{)}]\!] = [\![\texttt{(not }(subst_\beta a)\texttt{)}]\!] \tag{4.15}$$

$$subst_\beta [\![\texttt{(and }l_1\ \dots\ l_n\texttt{)}]\!] = [\![\texttt{(and }(subst_\beta l_1)\ \dots\ (subst_\beta l_n)\texttt{)}]\!] \tag{4.16}$$

where $\operatorname{dom} R$ denotes the domain of a relation R, or, more formally,

$$\operatorname{dom} R = \{x : X \mid \exists\, y : Y \bullet xRy\}$$

4.1.3 Worlds; Semantics

A *world* $W : World$ is a set of ground atoms: the facts that are true in this world. A formula f is *valid* in a world W, written as $W \models f$, if all positive literals in f hold and none of its negative literals. Formally:

$$W \models f \Leftrightarrow ((a \in f \Rightarrow a \in W) \wedge ([\![\texttt{(not }a\texttt{)}]\!] \in f \Rightarrow a \notin W)) \tag{4.17}$$

Note that we make use of the *closed world assumption*, a very economical way of modelling the world: All literals mentioned in W are true in the world, all literals not mentioned in W are false.

4.1.4 Operators

An *operator* ω : *Operator* is a pair of two ground formulas, the precondition formula p and the effect formula e.[3] We write this as ω = `(:precondition` p `:effect` e`)`. We use $pre(\omega)$ and $\mathit{eff}(\omega)$ to refer to the preconditions and effects of a given operator ω. An example is the following operator *CanvasUp*:[4]

$$\mathit{CanvasUp} = [\![\texttt{(:precondition (CanvasDown Canvas1)}\\ \qquad\qquad\texttt{:effect (not (CanvasDown Canvas1)))}]\!]$$

An operator ω is *executable* in a world W if its precondition formula holds. We write this as $Poss(\omega, W)$. Formally:

$$Poss(\omega, W) \Leftrightarrow W \models pre(\omega) \tag{4.18}$$

The result of executing an operator ω in a world W is a world W' where the effect formula of the operator is valid and nothing else has been changed.

There is a certain problem with effects: An effect such as $[\![$`(and` a `(not` a`))`$]\!]$ has no model and therefore can not be executed. We simply could abolish such contradictory effect formulas (and this would be certainly a good point from the declarative viewpoint) – however, with respect to the procedural semantics, we could argue that effects are executed sequentially, so the later effect should win. Such a procedural interpretation is specifically helpful when we consider a simple integration of first order effects in Section 5.1.

In order to capture this behavior, we define a function cff : *GroundFormula* $\longrightarrow$

[3]We will use the terms *precondition formula* and *preconditions* synonymously in the following. The same applies for *effect formula* and *effects*.

[4]Readers familiar with PDDL [Ghallab *et al.* 1998] will notice that our syntax is very similar to PDDL. Thus, they are in line with other descriptions in this thesis, which are often given in PDDL. However, here we use a simplified version rather than standard PDDL. For example, we usually omit the `:action` declaration in operators. This is to keep operator descriptions as concise as possible.

GroundFormula that gives the equivalent contradiction-free version of a formula:

$$cff\,[\![\texttt{()}]\!] = [\![\texttt{()}]\!] \tag{4.19}$$

$$cff(f \frown a) = (cff\,f \upharpoonright [\![\texttt{(not } a\texttt{)}]\!]) \frown a \tag{4.20}$$

$$cff(f \frown [\![\texttt{(not } a\texttt{)}]\!]) = (cff\,f \upharpoonright a) \frown [\![\texttt{(not } a\texttt{)}]\!] \tag{4.21}$$

Now the effect of an operator can be described by the function *Do* : *Operator* × *World* ⟶ *World* defined by

$$Poss(\omega, W) \Rightarrow Do(\omega, W) \models ceff\ \omega \tag{4.22}$$

$$\wedge\ \forall\, a : W \models a \wedge [\![\texttt{(not } a\texttt{)}]\!] \notin ceff\ \omega \Rightarrow Do(\omega, W) \models a \tag{4.23}$$

$$\wedge\ \forall\, a : W \not\models a \wedge a \notin ceff\ \omega \Rightarrow Do(\omega, W) \not\models a \tag{4.24}$$

where

$$ceff\ \omega = cff(eff\ \omega) \tag{4.25}$$

This is basically the definition used by the situation calculus in planning [Russell & Norvig 2010].

Alternatively, the operation of *Do* can be described by its procedural semantics:

$$Poss(\omega, W) \Rightarrow Do(\omega, W) = DoSeq(eff\ \omega, W) \tag{4.26}$$

where

$$DoSeq([\![\texttt{()}]\!], W) = W \tag{4.27}$$

$$DoSeq(a \frown f, W) = DoSeq(f, W \cup \{a\}) \tag{4.28}$$

$$DoSeq([\![\texttt{(not } a\texttt{)}]\!] \frown f, W) = DoSeq(f, W \setminus \{a\}) \tag{4.29}$$

4.2 Maes' Action Selection Algorithm

Having settled the necessary preliminaries, we are now in position to describe Maes' algorithm. We have given an informal description in Section 3.2.8. Here, we describe the algorithm more formally.

The key elements in Maes' algorithm are *competence modules*. Competence modules represent the actions of an intelligent agent. Formally, a competence module x is a tuple (ω, α) where ω is an operator as defined in Section 4.1.4. Furthermore, α denotes the competence module's activation level, which is a float value. In the following, we will often use ω_x and α_x to refer to a competence module x's operator and activation level.

The activation level α_x describes how likely competence module x is to become active. For competence module x to become active means that ω_x is executed. Thus, x can only become active if ω_x is *executable*, i.e. its precondition formula holds. More formally: $Poss(\omega_x, W)$. For reasons of simplicity, we will often speak about a competence module x's precondition formula when we actually mean ω_x's precondition formula. The same holds for effect formulas.

The competence modules are linked to one another in a network via three kinds of links: predecessor links *pl*, successor links *sl*, and conflicter links *cl*. There is a *predecessor link* from a competence module x to a competence module y for each of x's preconditions that equals one of y's effects. More formally:

$$pl(i,j) \Leftrightarrow i \in pre(x) \wedge j \in eff(y) \wedge i = j$$

If there is at least one predecessor link from x to y, we say that y is a *predecessor* of x or, more formally, $y \in Pred(x)$. Intuitively, y can "help" x to become active: When y becomes active, it fulfills one or more preconditions for x.

There is a *successor link* from a competence module x to a competence module y for every predecessor link from y to x. More formally:

$$sl(i,j) \Leftrightarrow pl(j,i)$$

If there is at least one successor link from x to y, we say that y is a *successor* of x or, more formally, $y \in Succ(x)$. Intuitively, y can "make use" of one or more propositions x has fulfilled.

Furthermore, there is a *conflicter link* from x to y for each of x's preconditions that is the opposite of one of y's effects. More formally:

$$cl(i,j) \Leftrightarrow i \in pre(x) \wedge j \in eff(y) \wedge i = \neg j$$

If there is at least one conflicter link from x to y, we say that y is a *conflicter* of x or, more formally, $y \in \mathit{Conf}(x)$. Intuitively, y "destroys" at least one precondition of x when executed.[5]

G is the set of *open goals*, i.e. the set of ground atoms the user wants the system to fulfill. R is the set of *protected subgoals*, i.e. goals that have been fulfilled by the algorithm. More formally, an open goal is an atom $g \in G$ with $g \notin R \wedge W \not\models g$. Note that both G and R can contain positive as well as negative literals (in contrast to W, which contains only positive literals).

Maes' algorithm runs in cycles: It successively selects competence modules which become active one after another in order to fulfill the goals. Whenever a goal has been fulfilled, it is transferred from the set of open goals to the set of protected subgoals. The algorithm stops when all goals are fulfilled. More formally: Assume that the algorithm starts at timestep 0. At timestep t, all goals are fulfilled and the algorithm stops. Hence, $G^{(0)}$ denotes the set of open goals when the algorithm starts. At timestep t, we then have the following state:

$$\begin{aligned}
&G^{(t)} = \emptyset \\
&R^{(t)} = G^{(0)} \\
&\forall g \in G^{(0)} : W^{(t)} \models g
\end{aligned}$$

Which competence module actually becomes active in a specific cycle is determined by spreading activation energy among the competence modules. How much energy a competence module x sends to a competence module y is determined by x's activation level and the links from x to y. Intuitively, the competence modules use their links to activate and inhibit each other, so that after a while energy accumulates in the modules that are most useful given the current situation and the open goals. Then the module with the highest activation level becomes active, provided that all of its preconditions are fulfilled and its activation level is above a certain threshold θ which can change in the course of the algorithm. In the beginning, θ

[5] *Links* in Maes' algorithm as well the AdDCo algorithm should not be confused with *causal links* in classical planning [Russell & Norvig 2003]. A causal link denotes that an action achieves a precondition for a subsequent action in a *plan*. In contrast, the links we introduce here do not connect actions in a plan, but nodes in the network that is used to *generate* the plan. Another difference are the relationships between nodes that the links express: While causal links in planning express a *predecessor* relation, links here can express the predecessor relation and two additional relations: the *successor* and the *conflicter* relation.

has the value of θ_0, the initial activation threshold. θ_0 is a global parameter the system designer has to specify. If no module becomes active, θ is lowered by a certain percentage – Maes suggests ten per cent here[6] – for the next cycle. If a module becomes active, θ is reset to θ_0.

Activation energy is injected into the network according to the current situation and the user's goals. The amount of energy injected is determined by a number of global parameters: In every cycle, each competence module receives energy for each fulfilled precondition. The intuitive idea is the following: The more fulfilled preconditions a competence module has, the closer it is to being executable, and hence, the more useful it is in the current situation. The amount of energy it receives for each fulfilled precondition is determined by the parameter ϕ. What "determined by" means in this context will be explained later. Furthermore, in every cycle, each competence module receives energy for each open goal that equals one of its effects. The intuitive idea is the following: The more open goals a competence module can fulfill, the quicker will all goals be fulfilled, and hence, the more desirable is it that this competence module becomes active. The amount of energy it receives for each open goal that equals one of its effects is determined by the parameter γ. Moreover, in every cycle, each competence module loses energy for each effect that is the opposite of a protected subgoal. The intuitive idea is the following: When becoming active, the competence module would undo one or more goals already achieved. Therefore it is not desirable that this competence module becomes active. The amount of energy it loses for each effect that is the opposite of a protected subgoal is determined by the parameter δ.

Another parameter is π, the mean level of activation for the competence modules in the network. After each cycle, a decay function is applied to each competence module's activation level in order to keep π constant. The parameters θ_0, π, ϕ, γ, and δ must be specified by the system designer. Example values are $\theta_0 = 45$, $\pi = 20$, $\phi = 20$, $\gamma = 50$, and $\delta = 40$.

These parameters also determine which amount of energy the competence modules receive according to their links. Each competence module that is not executable sends energy to each of its predecessors for every unfulfilled precondition. The amount of energy sent is determined by α, the competence module's activation level. Furthermore, each competence module that is executable sends energy to

[6]We have evaluated different percentages and found that ten per cent is indeed a good choice.

each of its successors for every effect that is not part of the current world state. The amount of energy sent is determined by $\alpha(\phi/\gamma)$. Furthermore, each competence module takes away energy from each of its conflicters for every fulfilled precondition. The amount of energy taken away is determined by $\alpha(\delta/\gamma)$.

We now describe what is meant by "determined by". Each input or removal of activation energy from competence module x is divided by two factors:

1. $| \, pre(x) \, |$, the number of literals in the precondition formula of x (in case x receives energy through a precondition, i.e. in case it receives energy from a predecessor or from the current situation due to a fulfilled precondition), or $| \, \mathit{eff}(x) \, |$, the number of literals in the effect formula of x (in case x receives energy through an effect, i.e. input coming from an open goal, or from a successor, or in case energy is taken away by a conflicter or by a protected subgoal), and
2. either one of m_i or a_i, where
   - m_i is the number of modules in the network that have literal i in their precondition formula (in case the module receives energy from a predecessor or from the current situation due to a fulfilled precondition),
   - a_i is the number of modules in the network that have literal i in their effect formula (in case the module receives energy from a successor or from an open goal, or in case the module loses energy due to a conflicter or due to a protected subgoal)

The intuitive idea of the first factor is to prevent competence modules with lots of preconditions/effects from being preferred over those with few, because they have more sources of activation energy. The intuitive idea of the second factor is to divide energy among those modules that have the same precondition fulfilled, that achieve the same goal or that can undo the same literal, in order to make them compete with one another to become active.

In the next section, we present Maes' action selection algorithm, which we call the *PM algorithm* (Pattie Maes' algorithm) in the following.

4.2.1 The PM Algorithm

The algorithm performs a loop. At every timestep, the following computation takes place for all competence modules. We describe it for timestep $t + 1$ and competence module x. Here, C denotes the set of competence modules in the network.

Furthermore, $\Xi^{(t)}$ denotes the set of executable competence modules at timestep t, or, more formally, $x \in \Xi^{(t)} \Leftrightarrow Poss(\omega_x, W^{(t)})$.

1. Compute new activation level of competence module x:

$$\begin{aligned}
\tilde{\alpha}_x^{(t+1)} = {} & \alpha_x^{(t)} \\
& + \sum_{i \in pre(x) | W^{(t)} \models i} \frac{\phi}{m_i |pre(x)|} \quad \textit{(activation for world state)} \\
& + \sum_{i \in \mathit{eff}(x) \cap G^{(t)}} \frac{\gamma}{a_i |\mathit{eff}(x)|} \quad \textit{(activation for open goals)} \\
& - \sum_{i \in \mathit{eff}(x) | \neg\ i \in R^{(t)}} \frac{\delta}{a_i |\mathit{eff}(x)|} \quad \textit{(inhibition for protected subgoals)} \\
& + \sum_{z \in Pred(x) \cap \Xi^{(t)}} \; \sum_{i \in (pre(x) \cap \mathit{eff}(z)) | W^{(t)} \not\models i} \frac{\alpha_z^{(t)} \phi}{\gamma m_i |pre(x)|} \quad \textit{(activation by predecessors)} \\
& + \sum_{z \in Succ(x) \cap \bar{\Xi}^{(t)}} \; \sum_{i \in (pre(z) \cap \mathit{eff}(x)) | W^{(t)} \not\models i} \frac{\alpha_z^{(t)}}{a_i |\mathit{eff}(x)|} \quad \textit{(activation by successors)} \\
& - \sum_{z \in Conf(x)} \; \sum_{\neg\ i \in pre(z) | i \in \mathit{eff}(x) | W^{(t)} \models \neg\ i} \frac{\alpha_z^{(t)} \delta}{\gamma a_i |\mathit{eff}(x)|} \quad \textit{(inhibition by conflicters)}
\end{aligned}$$

2. Compute overall activation energy in the network:

$$\tilde{\alpha}^{(t+1)} = \sum_{y \in C} \tilde{\alpha}_y^{(t+1)}$$

3. Normalize activation level for module x to keep total activation $\pi|C|$ constant:

$$\alpha_x^{(t+1)} = \frac{\tilde{\alpha}_x^{(t+1)}}{\tilde{\alpha}^{(t+1)}} \pi |C|$$

4. Compute set of execution candidates:

$$E^{(t+1)} = \{\, x \in \Xi^{(t+1)} \mid \alpha_x^{(t+1)} = \max_{z \in \Xi^{(t+1)}} \alpha_z^{(t+1)} \wedge \alpha_x^{(t+1)} > \theta \,\}$$

5.a If $E^{(t+1)} = \{x_0\}$, so we have a unique execution candidate: Execute operator:

$$
\begin{aligned}
W^{(t+1)} &= Do(\omega_{x_0}, W^{(t)}) \quad \textit{(new world state)} \\
G^{(t+1)} &= G^{(0)} \setminus \{i \mid W^{(t+1)} \models i\} \quad \textit{(open goals: goals that are not true now)} \\
R^{(t+1)} &= G^{(0)} \cap \{i \mid W^{(t+1)} \models i\} \quad \textit{(protected subgoals: goals that are true now)} \\
\theta &= \theta_0
\end{aligned}
$$

5.b Else

$$\theta = \theta * 0.9$$

Table 4.1: Scenario 4

initial state	Notebook1 hosts Document1
	Notebook2 hosts Document2
goal state	Document1 shown on Canvas3
	Document2 shown on Canvas1
optimal	Notebook1 maximizes Document1
action	Notebook2 maximizes Document2
sequence	Canvas3 is lowered
	Canvas1 is lowered
	Crossbar connects Notebook1 to Projector3
	Crossbar connects Notebook2 to Projector1
	Projector3 shows Document1 on Canvas3
	Projector1 shows Document2 on Canvas1

Table 4.2: Results

Scenario 4	
parameters	$\pi = 80$
	$\theta = 45$
	$\phi = 20$
	$\gamma = 50$
	$\delta = 40$
number of competence modules	154
number of cycles	22
length of action sequence	22

4.3 Maes' Algorithm in Smart Environments

The PM algorithm in its original form has two major drawbacks which make it unsuitable for the domain of smart ad-hoc environments:

- The first problem refers to the way the PM algorithm is implemented. The algorithm was designed for arbitrating between multiple behaviors of a *single* agent. Hence, the designer of such an agent develops a set of competence modules that represent the different behaviors of the agent. These behaviors could for instance be different arm movements of a robot. The modules are designed to reside and run on a single agent, and the PM algorithm does not provide for adding or removing competence modules at run-time. Furthermore, the interfaces of the competence modules, i.e. the preconditions and effects of their operators, must match exactly. Thus, one could say that the modules must be "hand-wired" by the designer. This implies that the PM algorithm cannot be applied for arbitrating between behaviors of *multiple* agents in distributed and dynamic settings, which may join and leave at run-time.
- The second problem refers to the action selection process itself. Maes tested the algorithm in rather small domains with few operators. A large number of operators can cause the algorithm to exhibit unpredictable behavior and the action selection process may produce several unnecessary actions. Consider Scenario 4 in Table 4.1.[7] This is a smart environment scenario we developed in accordance with the domain analysis in Chapter 2. It comprises 18 devices: eight canvasses, two notebooks, two documents, a video crossbar, four fixed projectors (Projector1 to Projector4) and a movable projector (Projector5). There are two goals: Document1 should be shown on Canvas3, and Document2 should be shown on Canvas1. The optimal action sequence consists of 8 actions. Table 4.2 lists the results obtained with the PM algorithm. The parameters were carefully hand-tuned: The results presented here are the best we obtained by running the algorithm with various different parameter settings.[8] The PM algorithm finds an action sequence consisting of 22 ac-

[7]The scenario in Table 4.1 is one of four scenarios we use for the evaluation of the AdDCo algorithm in Section 4.5. We describe these scenarios in a notation that is close to human language for reasons of clarity and understandability.

[8]Notice that hand-tuning the parameters is only necessary for the PM algorithm. For the AdDCo

tions. In other words, this sequence contains 13 unnecessary actions. Figure 4.1 depicts the corresponding activation levels of relevant competence modules throughout the run of the algorithm.[9] The thick red line at $y = 45$ depicts the activation threshold. Peaks with activation levels falling to 0 in the next cycle correspond to competence modules being selected and executed. One can see that the activation levels are always much higher than the activation threshold, and the activation threshold remains constant throughout the run of the algorithm. This indicates that the activation threshold has no influence on the action selection process. In every cycle, a competence module is chosen without much arbitration – the algorithm does not exhibit reasonable behavior. The problem here is that 154 competence modules are necessary to model this scenario.[10] Most of those 154 competence modules are not able to contribute to the goals, but nevertheless receive activation energy through their links. This results in the shown behavior.

In order to make Maes' algorithm applicable in smart ad-hoc environments, those two problems must be solved. In the following sections, we introduce the following adaptations:

1. We distribute the algorithm over the devices in a smart environment. This comprises the following changes:
    - We let the modules communicate via a network.
    - We let each module build up a partial world model containing representations of other devices and modules in the environment.
    - Instead of prewiring *operators*, we introduce templates (so-called *operator schemes*). These contain variables that correspond to device types. At run-time, these can be used to create operators by binding the variables to names of actual devices. Thus, the network need not be hand-wired anymore, but can be built up entirely at run-time.
    - We let the modules handle all synchronization issues themselves.
2. We reduce the network of modules at run-time to those that can actually

algorithm, a robust parameter setting exists that is feasible for a wide variety of scenarios. We discuss this issue further in Section 4.5.

[9]To prevent the figure from becoming too cluttered, only the activation levels of competence modules that are executed at some point are shown in this plot.

[10]In Section 4.5.6 we explain why Maes' algorithm requires this kind of modelling.

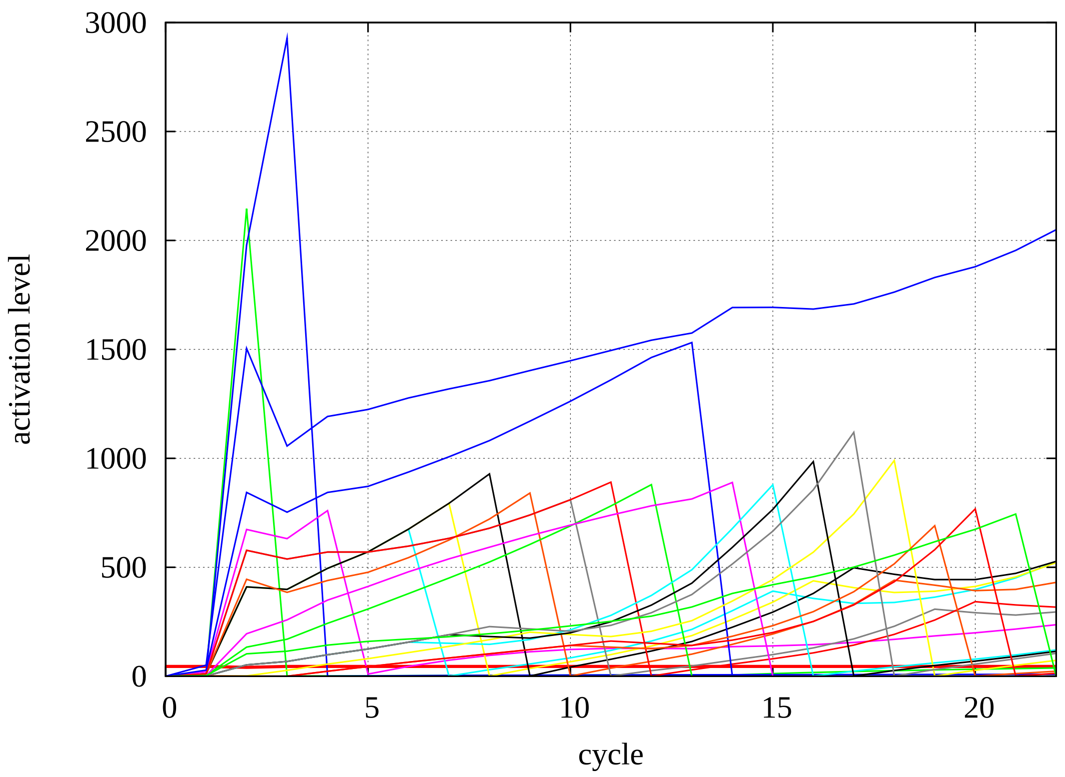


Figure 4.1: The activation flow in the PM algorithm in Scenario 4.

contribute to a goal.

We call our modified algorithm the *AdDCo* (Ad-hoc Device Cooperation) algorithm. With the changes described above, we are able to improve the behavior of the algorithm significantly. Scenario 4, for example, can now be modeled with only 32 operators, and the AdDCo algorithm finds an action sequence consisting of 9 actions. The resulting activation pattern is shown in Figure 4.2.[11] A detailed comparison of the PM algorithm and the AdDCo algorithm follows in Section 4.5. In the following, we describe the modifications in detail.

4.4 The AdDCo Algorithm

In this section, the AdDCo algorithm is described. We first settle the preliminaries, i.e. we explain the overall system architecture and describe the assumptions that

[11] Again, only the activation levels of competence modules that are executed at some point are shown in this plot.

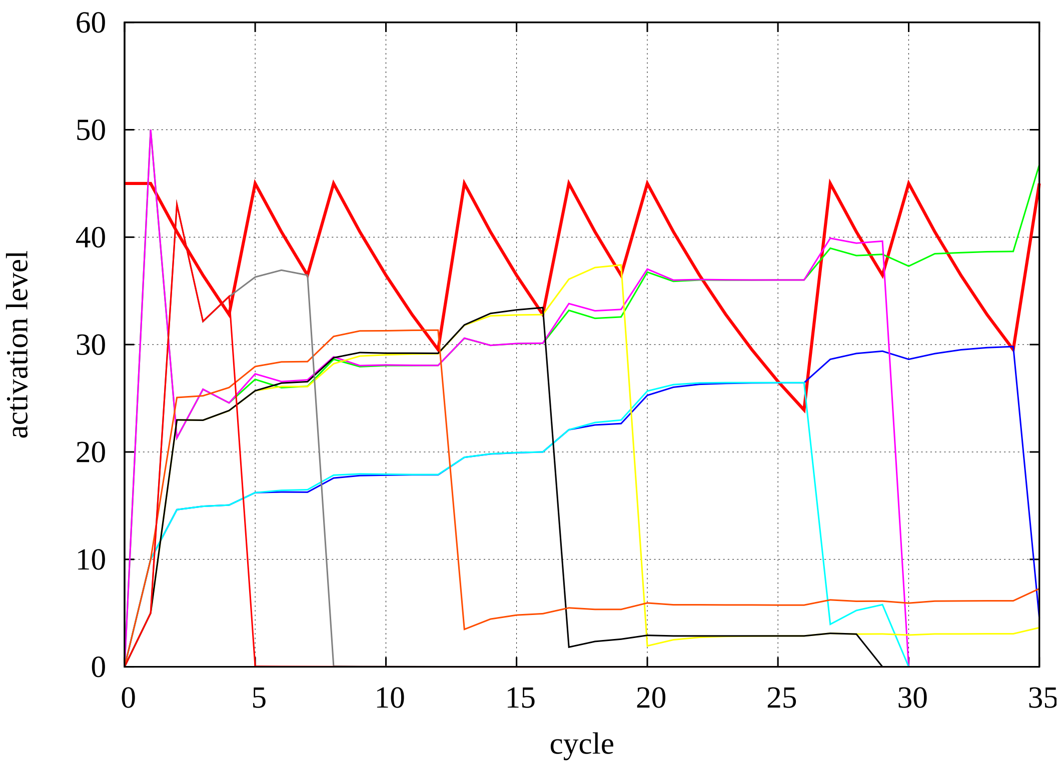


Figure 4.2: The activation flow in the AdDCo algorithm in Scenario 4.

are necessary for the algorithm to work. We then describe the algorithm in detail. This description comprises two parts: network maintenance and action selection.

4.4.1 The Overall System Architecture

The overall system architecture is depicted in Figure 4.3. We assume that all devices are connected to a common network and have elementary computing capabilities and some memory. For reasons of simplicity we regard entities in a broader sense as devices, too, e.g. documents. Of course a document does not have a processor, nor does it have memory or network access. Thus, we assume that for such devices a surrogate computer will provide the required functionality. In the case of a document, this could for instance be the notebook that hosts the document. Two types of software components run on each device: *CompMods* and *ECo services*. In Figure 4.3, CompMods are depicted as white rectangles with a black border, while ECo services are depicted as red rectangles.

CompMods are the core components of the AdDCo algorithm. There is a

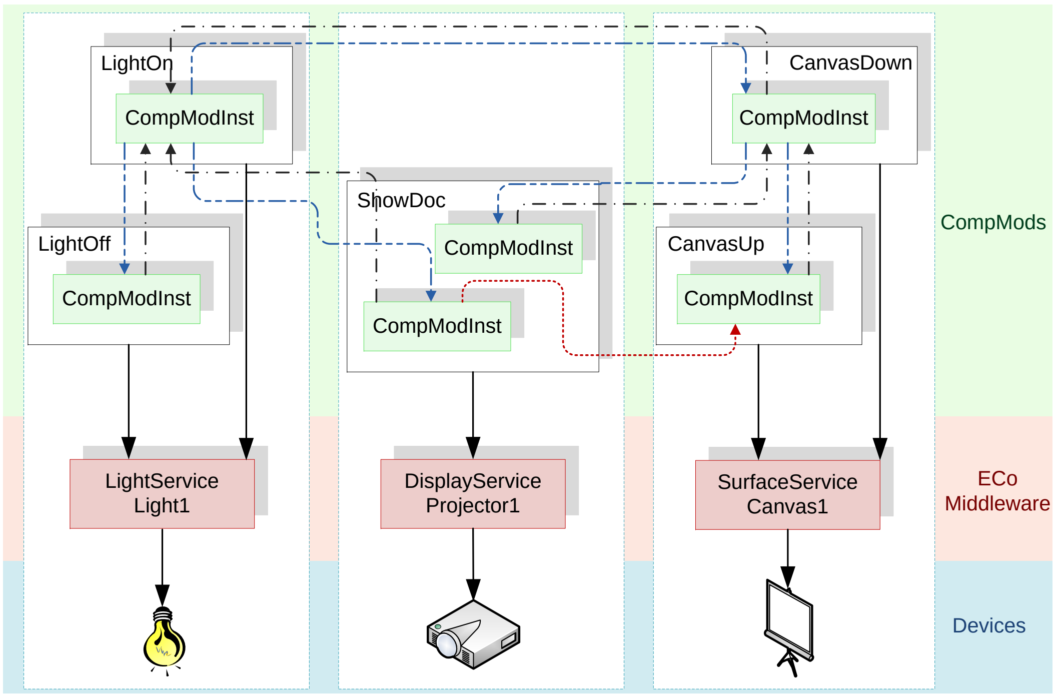


Figure 4.3: The overall system architecture.

CompMod for each action a device can perform. Each CompMod carries a declarative description of its assigned action, called an *operator scheme*. Operator schemes are described in terms of *preconditions* and *effects*. These may contain variables, each of which stands for a device type. At run-time, these variables are bound to names of devices of that type which are present in the ensemble. With each such binding, an *operator* is generated. This process is described in a detailed and formal way in Section 4.4.5. An operator plus some additional data structures forms a *CompModInst* (*CMI*). In Figure 4.3, CompModInsts are depicted as green rectangles. In the AdDCo algorithm, CompModInsts are the equivalent to the competence modules in the PM algorithm.

The purpose of the CompMods is to collectively decide which action should be executed when. To this end, each CompMod can perform simple calculations, has memory and communicates with the other CompMods via the network. The CompModInsts are connected by links according to their preconditions and effects, just as the competence modules in the PM algorithm.[12] Figure 4.4 depicts four CompMod-

[12]Note that we often speak about a CompModInst's preconditions and effects although, strictly speaking, we mean the preconditions and effects of its operator.

Insts with preconditions and effects described in PDDL [Ghallab *et al.* 1998]. The black, blue and red dotted arrows represent predecessor, successor and conflicter links. The yellow ellipsoids above the CompModInsts represent the percepts[13] – the literals that are part of the current world state, while the yellow ellipsoids below the CompModInsts represent the open goals. The yellow dotted arrows show the energy the CompModInsts receive from percepts and goals. The process of action selection is described in detail in Section 4.4.11. Whenever there is an open goal, the CompMods communicate with one another to select CompModInsts for execution in order to fulfill this goal. This selection is based on energy distribution according to the links, similar to the PM algorithm. When a CompModInst has been chosen, its CompMod sends a command to the ECo middleware.

The *ECo middleware*, developed by Heider and Giersich [Heider 2010, Giersich 2010], consists of *ECo services*. ECo stands for *Ensemble Communication Framework*. The ECo services are distributed across the devices: There is an ECo service on each device. Communication among the ECo services is based on multicast. The purpose of an ECo service is to translate the high-level commands coming from the CompMods into a low-level command that directly drives the device. Then, the action chosen by the ensemble of CompMods is executed in the environment.

4.4.2 The Architecture of a CompMod

As said before, CompMods form the core of the AdDCo algorithm. This section gives an overview of the algorithms carried out within a CompMod and the data structures each CompMod stores. These components are also depicted in Figure 4.5. Details follow in the forthcoming sections.

A CompMod consists of four components:

- a *world model* that contains information about the CompMod itself and the rest of the ensemble
- a *communication component* that sends messages to the rest of the ensemble and handles incoming messages
- a component for *network maintenance* that builds up and maintains the world

[13]Note that according to (4.17), the world state does not contain any negative literals due to the closed world assumption. However, in Figure 4.4 we include the negative literal `(not (CanvasDown Canvas1))` to show its relation to *CanvasDown*'s precondition.

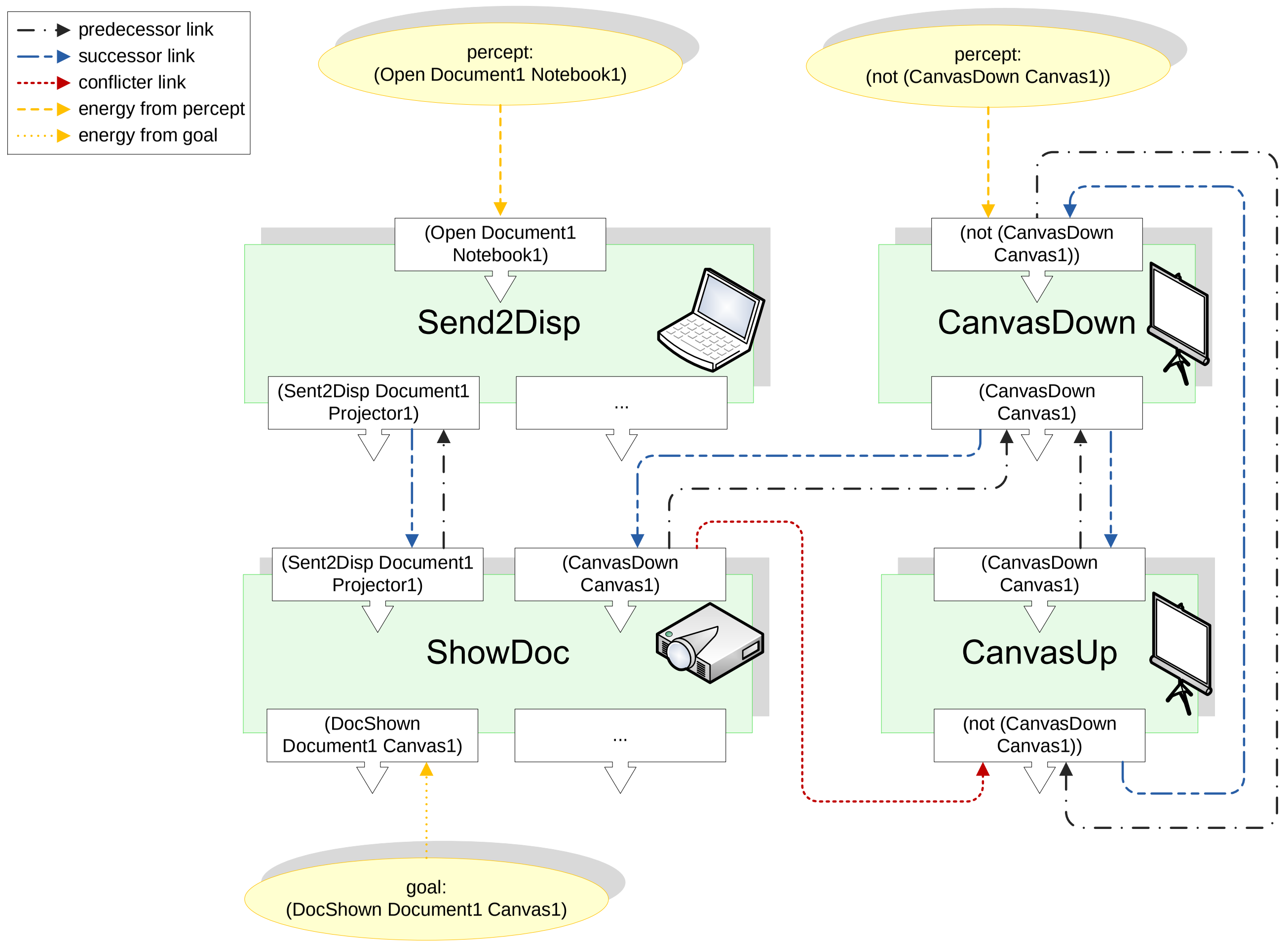

predecessor link
successor link
conflicter link
energy from percept
energy from goal
percept:
(Open Document1 Notebook1)
percept:
(not (CanvasDown Canvas1))
(Open Document1 Notebook1)
Send2Disp
(Sent2Disp Document1 Projector1)
...
(not (CanvasDown Canvas1))
CanvasDown
(CanvasDown Canvas1)
(Sent2Disp Document1 Projector1)
(CanvasDown Canvas1)
ShowDoc
(DocShown Document1 Canvas1)
...
(CanvasDown Canvas1)
CanvasUp
(not (CanvasDown Canvas1))
goal:
(DocShown Document1 Canvas1)

Figure 4.4: Four CompModInsts connected by links. Additionally, the flow of energy from percepts and goals to CompModInsts is depicted.

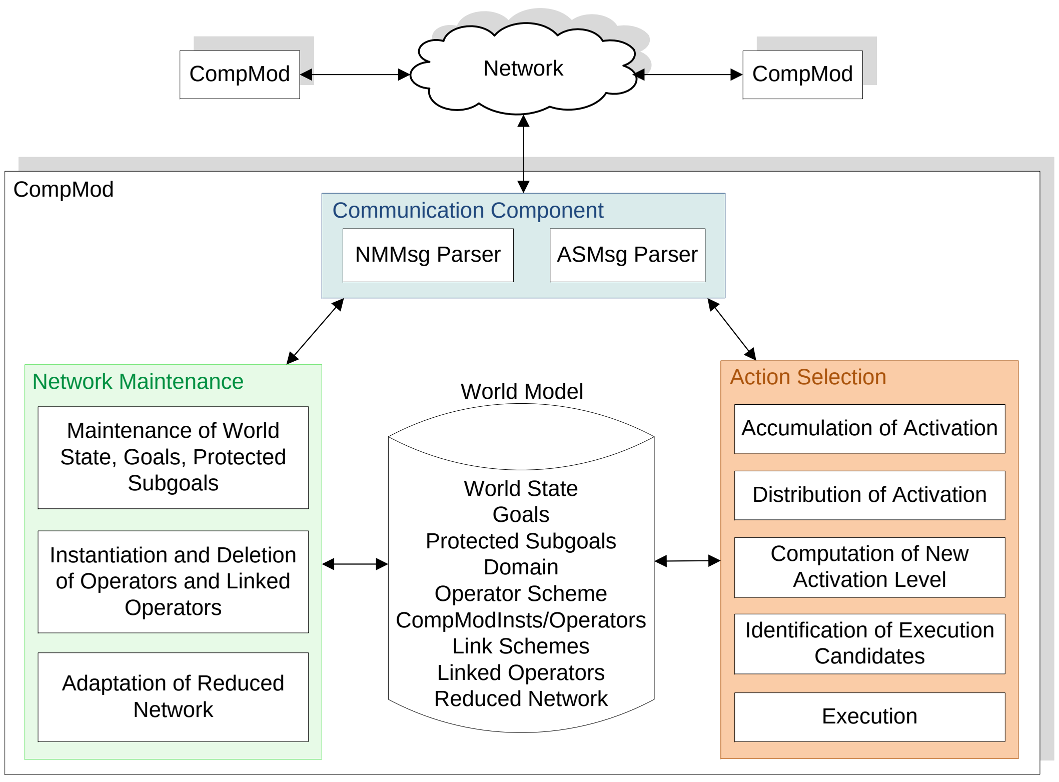

CompMod
Network
CompMod
CompMod
Communication Component
NMMsg Parser
ASMsg Parser
Network Maintenance
Maintenance of World State, Goals, Protected Subgoals
Instantiation and Deletion of Operators and Linked Operators
Adaptation of Reduced Network
World Model
World State
Goals
Protected Subgoals
Domain
Operator Scheme
CompModInsts/Operators
Link Schemes
Linked Operators
Reduced Network
Action Selection
Accumulation of Activation
Distribution of Activation
Computation of New Activation Level
Identification of Execution Candidates
Execution

Figure 4.5: The architecture of a CompMod.

model

- a component for *action selection* that enables the CompMod to take part in the distributed decision-making process

The *world model* is not a complete model of the CompMod's surroundings, but contains only the information necessary for the respective CompMod. The world model comprises the following information:

- The current *world state*: It consists of literals that we call *percepts*. These literals are delivered to the CompMods by hardware or software sensors. An example is the literal `(Open Document1 Notebook1)`.
- The open *goals*: These are literals that are not yet true in the environment, but which are to be fulfilled by the CompMods. For the AdDCo algorithm to work, we assume that the goals are correctly provided by the *intention analysis*, as described in Section 1.2.
- The *protected subgoals*: These are literals that originally were open goals, but have been fulfilled by the CompMods in the course of the algorithm.

Hence they are now part of the current world state.

- The *domain*: The domain describes all devices currently in the ensemble. It maps device types to their names. We introduce it formally in Section 4.4.5.
- The CompMod's *operator scheme*: This is a description of the action assigned to the CompMod. It consists of a *precondition formula* and an *effect formula*. We introduce it formally in Section 4.4.5. The operator scheme is the only element of the world model that cannot be changed in the course of the AdDCo algorithm.
- The CompMod's *CompModInsts*: A CompModInst consists of an operator which represents a fully specified action that can be executed in the environment. In addition, the CompModInst stores data required for the action selection algorithm, e.g. its activation level.
- *Link Schemes*: Link schemes are operator schemes of other CompMods in the ensemble the CompMod is linked to. This is detailed in Section 4.4.6.
- *Linked operators*: These are operators that are linked to the operators of the CompMod. The CompMod instantiates link schemes into linked operators using the domain. Details follow in Section 4.4.6.
- The *reduced network*: The reduced network contains those CompModInsts that can contribute to the fulfillment of an open goal. Each CompMod stores which of its CompModInsts is part of the reduced network. This is described in detail in Section 4.4.8.

The *communication component* handles all communication between the CompMod and the rest of the ensemble by sending and receiving messages. Each message is either a *network maintenance message* (denoted by *NMMsg* in Figure 4.5) or an *action selection message* (denoted by *ASMsg* in Figure 4.5). Both are in the format of a *message vector*. We explain this in Section 4.4.3. Incoming messages are parsed and distributed to the network maintenance component and the action selection component. There is a separate parser for network maintenance messages and action selection messages. Details follow in Section 4.4.4.

When the CompMod enters the ensemble, the component for *network maintenance* builds up the world model. Afterwards, it ensures that the world model is always up to date. To this end, the network maintenance component is called by the Communication Component whenever a network maintenance message (NMMsg) is received. Network maintenance is described in detail in Sections 4.4.5, 4.4.6, 4.4.7, 4.4.8, and 4.4.9.

All CompMods in the ensemble collectively search for an action sequence that fulfills the user's goals. To this end, each of them executes an *action selection* algorithm that runs in cycles. At the end of each cycle, an action can be executed. Each cycle consists of a series of computation and communication steps. In each step, the CompMods compute small partial solutions, then exchange these solutions via sending and receiving messages. Using the information received, each CompMod then computes the information needed for the next step and so on. The action selection algorithm is described in Section 4.4.11. The computation part is similar to the PM algorithm, albeit there are some important differences which we discuss in Sections 4.4.10 and 4.4.12. The communication part, however, has no equivalent in the PM algorithm. It is due to the fact that the AdDCo algorithm is a distributed algorithm.

4.4.3 The Message Vectors

As the AdDCo algorithm is a distributed algorithm, the CompMods communicate to build up their world model, keep it up to date and cooperatively manage the process of action selection. All communication can be described by two message vectors that are broadcast by a CompMod for the purpose of sharing information with other CompMods: the network maintenance vector and the action selection vector. Furthermore, the network maintenance vector is also used by the intention analysis and sensors to announce a new goal or a percept and by a device announcing its joining or leaving the ensemble. The action selection vector has 7 elements and is depicted in Table 4.3; the network maintenance vector has 13 elements and is depicted in Table 4.4.

1	$\omega_x(r)$	operator of a CMI x that is the receiver of 2, 3, and 4
2	$\alpha_{zx}^{(t+1)}(pred)$	activation sent to 1 by a successor
3	$\alpha_{zx}^{(t+1)}(succ)$	activation sent to 1 by a predecessor
4	$\alpha_{zx}^{(t+1)}(conf)$	inhibition sent to 1 by a conflicter
5	$\omega_x(s)$	operator of a CMI x that is the sender of 6 or 7
6	$\alpha_x^{(t+1)}(a)$	5's preliminary activation level
7	$\widetilde{\varepsilon_x}$	5's executability

Table 4.3: The action selection message vector.

1	W	current world state
2	w	new literal (percept) to be included in the world state
3	G	currently open goals
4	g	new open goal
5	R	set of current protected subgoals
6	σ_a	operator scheme of a new CompMod a
7	σ_o	operator scheme of a CompMod o that is not new
8	σ_l	operator scheme of a CompMod l leaving the ensemble
9	τ_a	description of a device joining the ensemble
10	τ	current domain (descriptions of all devices in the ensemble)
11	τ_l	description of a device leaving the ensemble
12	$\omega_x(a)$	operator of a CMI x that is added to reduced network
13	$\omega_x(l)$	operator of a CMI x that is deleted from reduced network

Table 4.4: The network maintenance message vector.

Here, we will not explain the meaning of each of the elements. This will become clear in the next sections, when we introduce operator schemes, link schemes etc. For now, it is only important to know that each element triggers a certain reaction in the CompMods that receive the message. This, too, is later described in detail. In an actual message vector usually only a subset of the 13 elements of the network maintenance vector or of the 7 elements of the action selection vector actually contains information, while the other elements are set to NULL. For example, when a new CompMod a joins the ensemble, it informs the other CompMods about its presence by sending its operator scheme σ_a in a network maintenance message of the form NMMsg(NULL, NULL, NULL, NULL, NULL, σ_a, NULL, NULL, NULL, NULL, NULL, NULL, NULL). Network maintenance messages start with *NMMsg*, while action selection messages start with *ASMsg*. For better readability, we will often write messages in a form that includes only the non-NULL elements preceded by their position in the vector. Thus, our example message reads as NMMsg(6: σ_a).

When a CompMod receives a network maintenance message, it must adapt its world model according to the message. This may cause inconsistencies if action selection continues during adaptation. Therefore, action selection is automatically stopped when a network maintenance message is recognized. When the world model has been adapted, action selection is restarted.

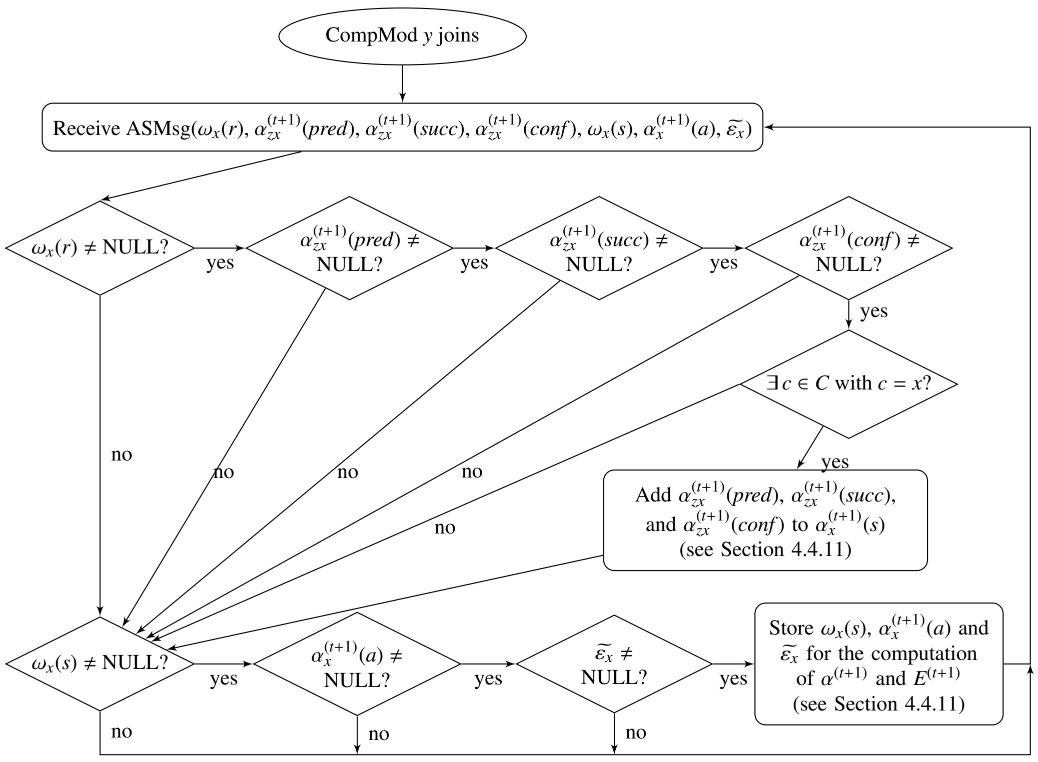

CompMod y joins
Receive ASMsg($\omega_x(r)$, $\alpha_{zx}^{(t+1)}(pred)$, $\alpha_{zx}^{(t+1)}(succ)$, $\alpha_{zx}^{(t+1)}(conf)$, $\omega_x(s)$, $\alpha_x^{(t+1)}(a)$, $\widetilde{\varepsilon_x}$)
$\omega_x(r) \neq$ NULL?
$\alpha_{zx}^{(t+1)}(pred) \neq$ NULL?
$\alpha_{zx}^{(t+1)}(succ) \neq$ NULL?
$\alpha_{zx}^{(t+1)}(conf) \neq$ NULL?
$\exists c \in C$ with $c = x$?
Add $\alpha_{zx}^{(t+1)}(pred)$, $\alpha_{zx}^{(t+1)}(succ)$, and $\alpha_{zx}^{(t+1)}(conf)$ to $\alpha_x^{(t+1)}(s)$ (see Section 4.4.11)
$\omega_x(s) \neq$ NULL?
$\alpha_x^{(t+1)}(a) \neq$ NULL?
$\widetilde{\varepsilon_x} \neq$ NULL?
Store $\omega_x(s)$, $\alpha_x^{(t+1)}(a)$ and $\widetilde{\varepsilon_x}$ for the computation of $\alpha^{(t+1)}$ and $E^{(t+1)}$ (see Section 4.4.11)
yes
no

Figure 4.6: Action selection messages parsed and distributed by the Communication Component.

4.4.4 Parsing the Message Vectors

Both the network maintenance vector and the action selection vector are parsed by the Communication Component. According to the elements the vectors contain, certain actions are triggered. This is depicted in the flowchart in Figure 4.6 for the action selection vector and in the flowchart in Figure 4.7 for the network maintenance vector. The flowcharts are intended to give the reader the overall picture of inter-CompMod communication while the details will be explained in the forthcoming sections. Hence, the flowcharts are best skimmed at first reading and later viewed again when the details have been explained.

As the AdDCo algorithm is a distributed algorithm, each CompMod has a Communication Component and hence, each CompMod parses network maintenance messages and action selection messages. The flowcharts describe this for CompMod y. CompMod y's variables are marked by the index y – for example, σ_y denotes y's operator scheme. Furthermore, C denotes the set of y's CompModInsts.

In the following sections, we describe network maintenance and action selec-

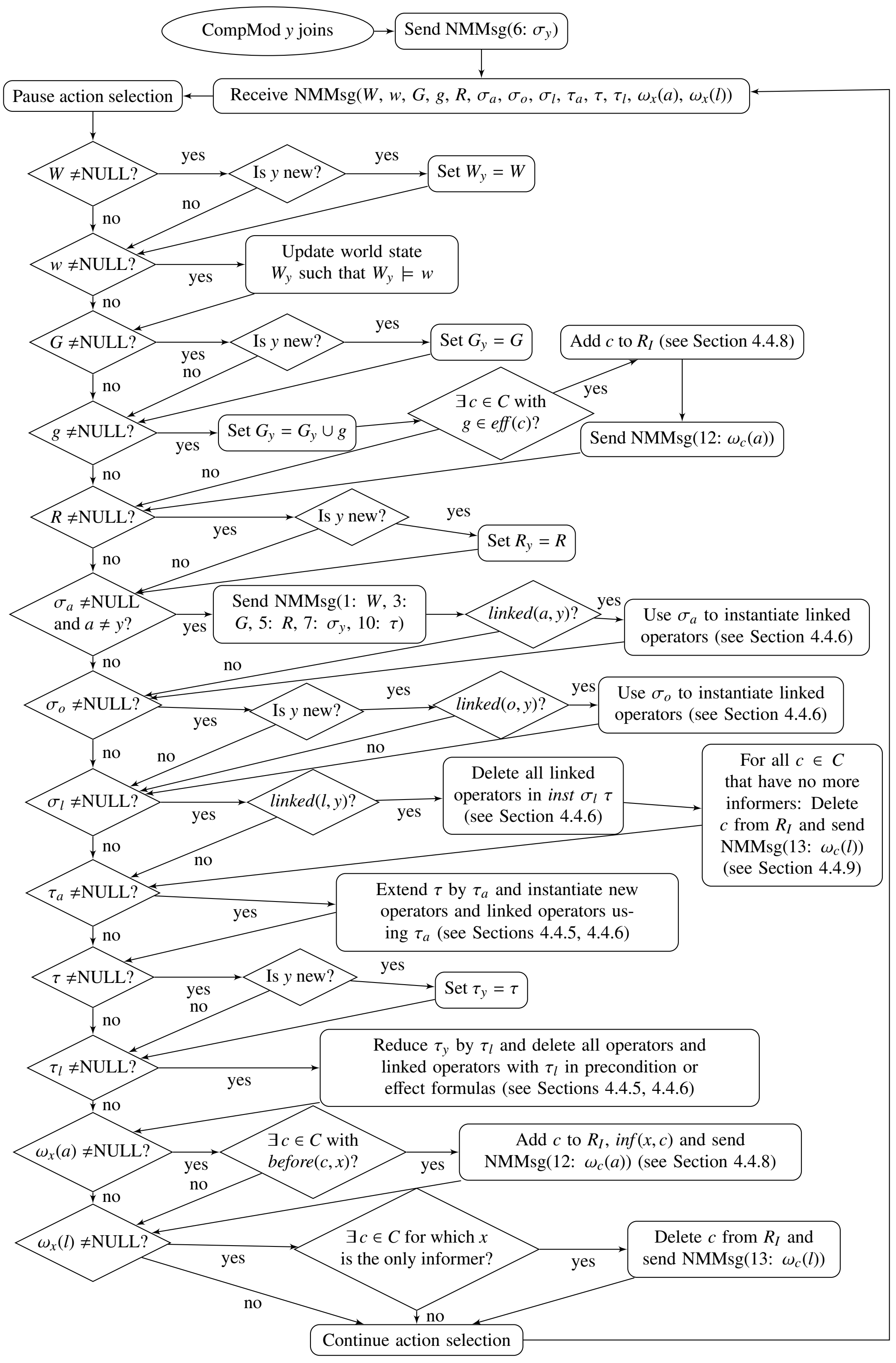

CompMod y joins
Send NMMsg(6: σ_y)
Pause action selection
Receive NMMsg(W, w, G, g, R, σ_a, σ_o, σ_l, τ_a, τ, τ_l, $\omega_x(a)$, $\omega_x(l)$)
W ≠NULL?
Is y new?
Set $W_y = W$
w ≠NULL?
Update world state W_y such that $W_y \models w$
G ≠NULL?
Is y new?
Set $G_y = G$
Add c to R_I (see Section 4.4.8)
g ≠NULL?
Set $G_y = G_y \cup g$
$\exists c \in C$ with $g \in eff(c)$?
Send NMMsg(12: $\omega_c(a)$)
R ≠NULL?
Is y new?
Set $R_y = R$
σ_a ≠NULL and $a \neq y$?
Send NMMsg(1: W, 3: G, 5: R, 7: σ_y, 10: τ)
$linked(a, y)$?
Use σ_a to instantiate linked operators (see Section 4.4.6)
σ_o ≠NULL?
Is y new?
$linked(o, y)$?
Use σ_o to instantiate linked operators (see Section 4.4.6)
σ_l ≠NULL?
$linked(l, y)$?
Delete all linked operators in $inst\ \sigma_l\ \tau$ (see Section 4.4.6)
For all $c \in C$ that have no more informers: Delete c from R_I and send NMMsg(13: $\omega_c(l)$) (see Section 4.4.9)
τ_a ≠NULL?
Extend τ by τ_a and instantiate new operators and linked operators using τ_a (see Sections 4.4.5, 4.4.6)
τ ≠NULL?
Is y new?
Set $\tau_y = \tau$
τ_l ≠NULL?
Reduce τ_y by τ_l and delete all operators and linked operators with τ_l in precondition or effect formulas (see Sections 4.4.5, 4.4.6)
$\omega_x(a)$ ≠NULL?
$\exists c \in C$ with $before(c, x)$?
Add c to R_I, $inf(x, c)$ and send NMMsg(12: $\omega_c(a)$) (see Section 4.4.8)
$\omega_x(l)$ ≠NULL?
$\exists c \in C$ for which x is the only informer?
Delete c from R_I and send NMMsg(13: $\omega_c(l)$)
Continue action selection
yes
no

Figure 4.7: Network maintenance messages parsed and distributed by the Communication Component.

tion in detail.

4.4.5 Operator Schemes and Instantiation

In this section, we introduce operator schemes. Operator schemes do not exist in the PM algorithm. They bring the flexibility that enables the AdDCo algorithm to perform distributed run-time strategy synthesis. Informally, one can say that operator schemes are templates that can be instantiated into operators at run-time. A more formal definition follows.

In addition to the sets *Const*, *Var*, and *Predicate* defined in Section 4.1.1, we now introduce the set of type names, $d \in \mathit{Type}$, e.g. `Projector`, `Canvas`. Type names occur in *declarations*:

A *declaration* $\delta : \mathit{Decl}$ where $\mathit{Decl} = \mathit{Var} \longrightarrow \mathit{Type}$ is a finite map from variable names to type names. A *domain* $\tau : \mathit{Dom}$ where $\mathit{Dom} = \mathit{Type} \longrightarrow \mathbb{P}\,\mathit{Const}$ is a finite map from type names to sets of constants. The function *bindings* $: \mathit{Decl} \longrightarrow \mathit{Dom} \longrightarrow \mathbb{P}\,\mathit{Binding}$ gives all possible bindings for a given variable declaration and a domain

$$\mathit{bindings}\ \delta\ \tau = \{\, \beta : \mathit{Binding} \mid \mathrm{dom}\,\beta = \mathrm{dom}\,\delta \wedge \forall\, v \in \mathrm{dom}\,\delta : \beta\ v \in \tau(\delta\ v) \,\} \tag{4.30}$$

An *operator scheme* $\sigma : \mathit{OpScheme}$ where $\mathit{OpScheme} = \mathit{Decl} \times \mathit{Formula} \times \mathit{Formula}$ is a triple consisting of a declaration δ, a precondition formula p and an effect formula e. We write this $\sigma = [\![$`(:parameters` δ `:precondition` p `:effect` e`)`$]\!]$; we write $\mathit{decl}(\sigma)$ to denote the declaration of a given operator scheme σ, and we use $\mathit{pre}(\sigma)$ and $\mathit{eff}(\sigma)$ to refer to σ's precondition and effect formula, respectively. An example for an operator scheme is the following:

$$\begin{aligned} \sigma = \mathit{CanvasUp} = [\![& \texttt{(:parameters (?c - Canvas)} && (4.31)\\ & \texttt{:precondition (CanvasDown ?c)} && (4.32)\\ & \texttt{:effect (not (CanvasDown ?c)))}]\!] && (4.33) \end{aligned}$$

so that

$$decl(\sigma) = \{[\![\texttt{?c}]\!] \mapsto [\![\texttt{Canvas}]\!]\} \tag{4.34}$$

$$pre(\sigma) = [\![\texttt{(CanvasDown ?c)}]\!] \tag{4.35}$$

$$eff(\sigma) = [\![\texttt{(not (CanvasDown ?c))}]\!] \tag{4.36}$$

Instantiating an operator scheme σ with respect to a given domain τ amounts to computing all possible bindings for σ's declarations using all possible objects in τ and then creating all operators by substituting the binding in σ's precondition and effect formulas. It is given by the function $inst : OpScheme \longrightarrow Dom \longrightarrow \mathbb{P}\ Operator$, defined as follows:

$$\begin{aligned} inst\ \sigma\ \tau = \{\ [\![\texttt{(:precondition}\ p\ \texttt{:effect}\ e\texttt{)}]\!] \mid \exists \beta \in bindings\ (decl(\sigma))\ \tau :\\ p = subst_\beta(pre(\sigma))\\ \wedge\ e = subst_\beta(eff(\sigma))\ \} \end{aligned} \tag{4.37}$$

As an example, suppose we have the domain $\tau = \{Canvas \mapsto \{Canvas1, Canvas2\}\}$. Instantiating the operator scheme *CanvasUp* described above with respect to this domain yields the following two operators:

$$\begin{aligned} CanvasUpCanvas1 = [\![&\texttt{(:precondition (CanvasDown Canvas1)}\\ &\texttt{:effect (not (CanvasDown Canvas1)))}]\!] \end{aligned}$$

$$\begin{aligned} CanvasUpCanvas2 = [\![&\texttt{(:precondition (CanvasDown Canvas2)}\\ &\texttt{:effect (not (CanvasDown Canvas2)))}]\!] \end{aligned}$$

A more comprehensive example of instantiating an operator scheme can be found in Section 9.2 in the Appendix. Note that for every new operator that is instantiated, a CompModInst is generated. As said before, such a CompModInst stores not only the operator, but also additional data structures which are essential for the action selection algorithm.

Please also note that in the AdDCo algorithm, domains can change at run-time. This is due to the fact that semantically, a domain is a mapping from device types to device names. Thus, when a new device enters the ensemble, the domain τ is extended by a mapping τ_a. The new device announces its joining to the ensemble by sending a message NMMsg(9: τ_a). All CompMods receive this message and extend their domains accordingly. Consider adding $\tau_a = \{Canvas \mapsto \{Canvas3\}\}$ to the above example domain. This yields the extended domain $\{Canvas \mapsto \{Canvas1, Canvas2, Canvas3\}\}$. When a device joins, new operators are instantiated according to the definition above, and for each new operator a new CompModInst is generated. Likewise, τ_l denotes a device that leaves the ensemble, announced by a message NMMsg(11: τ_l). This causes the domain τ to be reduced by τ_l. Furthermore, it causes each CompModInst to be deleted whose operator contains the device name in its precondition or effect formula.

4.4.6 Link Schemes and Linked Operators

In Section 4.2, we have defined links for the competence modules in the PM algorithm. In analogy, we now define links for CompMods and CompModInsts in the AdDCo algorithm. The following definitions are valid if x and y are either both CompMods or if they are both CompModInsts. Notice that for reasons of simplicity we write $pre(x)$ and $eff(x)$ although, strictly speaking, this is not correct: We obviously do not mean x's precondition or effect formula, but the precondition/effect formula in x's operator (in case x is a CompModInst) or operator scheme (in case x is a CompMod), respectively.

We say that x is *linked* to y if y is a predecessor, successor, or conflicter of x:

$$linked(x, y) \Leftrightarrow y \in Pred(x) \lor y \in Succ(x) \lor y \in Conf(x)$$

We say that y is a predecessor of x (or $y \in Pred(x)$) if there is at least one predecessor link from x to y. There is a *predecessor link* from x to y for each of x's preconditions that equals one of y's effects. More formally:

$$pl(i,j) \Leftrightarrow i \in pre(x) \land j \in eff(y) \land i = j$$

We say that y is a *successor* of x (or $y \in Succ(x)$) if there is at least one successor link from x to y. There is a successor link from x to y for every predecessor link from y to x. More formally:

$$sl(i,j) \Leftrightarrow pl(j,i)$$

We say that y is a *conflicter* of x (or $y \in Conf(x)$) if there is at least one conflicter link from x to y. There is a conflicter link from x to y for each of x's preconditions that is the opposite of one of y's effects. More formally:

$$cl(i,j) \Leftrightarrow i \in pre(x) \wedge j \in eff(y) \wedge i = \neg j$$

Via the links, the CompModInsts form a network. This network is stored in a distributed fashion: Whenever a new CompMod enters the ensemble, it broadcasts its operator scheme in a message NMMsg(6: σ_a). Each CompMod then checks whether it is linked to the new CompMod. If yes, it stores the new CompMod's operator scheme, which we now call a *link scheme*. The CompMod now checks if it can instantiate the link scheme into *linked operators* using the domain τ. A CompMod instantiates linked operators in the same way as instantiating its own operators. We have described this process in Section 4.4.5. Furthermore, each of the other CompMods answers the new CompMod's request by sending its operator scheme along with some more information, which is detailed in the next section. This way, the new CompMod can build up an internal model of the CompMods it is linked to by instantiating linked operators. Instantiation of linked operators is repeated everytime a new device enters, i.e. whenever τ is extended by a new mapping τ_a. Hence, each CompMod stores exactly the part of the network relevant for itself: its own operators and the operators that are its predecessors, successors, and conflicters.

The reason why we let each CompMod instantiate all its linked operators itself instead of sending complete operators over the network is to reduce the number of messages that have to be sent. In Section 4.4.5 we have seen how operators are created for an operator scheme σ and a domain τ: For each binding $\beta \in bindings\ (decl(\sigma))\ \tau$, an operator is created. Each of those operators is, of course, potentially a linked operator to some other CompMod in the ensemble. This implies that if CompMods did not instantiate their linked operators themselves, for each β a message would have to be broadcast over the network, announcing the instantiation of a potential linked operator. This would lead to an increased message load. Of course, one could argue that when joining the ensemble, each CompMod could send all its operators in a bulk in a single message. But even then the message load would be significantly increased. This is due to the fact that the ensemble structure may change at any time. Consider that each CompMod needs to instantiate operators when a device joins and delete operators when a device leaves. In each of those cases, it would have to send an update message telling the other CompMods which operators have just been instantiated or deleted. Hence, it is more economical in terms of message load to just let each CompMod send its operator scheme σ and have the CompMods that are linked to it instantiate and delete their linked operators themselves.

When a CompMod leaves the network, it sends a message NMMsg(8: σ_l). Each CompMod now checks whether it is linked to the CompMod that leaves. In this case, it deletes the link scheme σ_l and all linked operators it has instantiated using σ_l. A similar process takes place whenever τ is reduced by τ_l because a device leaves: Then each CompMod deletes all linked operators that contain τ_l in their precondition or effect formula.

4.4.7 Building up the World Model

When a CompMod enters the ensemble, its world model is empty. It is built up via communication in the following way: Each new CompMod broadcasts its operator scheme in a message NMMsg(6: σ_a). This message serves two purposes: First, it tells the other CompMods that there is a new member in the ensemble that they must incorporate into their own world models, as described in Section 4.4.6. Second, it serves as a request to the other CompMods to send the new CompMod all relevant information about the current context. Each CompMod answers the request message with a message NMMsg(1: W, 3: G, 5: R, 7: σ_o, 10: τ). This is depicted in Figure 4.7. In detail, this message contains the following information:

- W, the current world state,
- G, the goals currently open,
- R, the current protected subgoals,
- σ_o, the sender's operator scheme,
- τ, the current domain (i.e. descriptions of all devices currently in the ensemble).

This information is then received by the new CompMod and used to build up its world model. Although all other CompMods receive the same message, it is only relevant for a new CompMod because existing CompMods have already built up their world model. Hence, the positions 1, 3, 5, 7, and 10 in the network maintenance vector are only parsed by new CompMods.

4.4.8 Adding CompModInsts to the Reduced Network

In a typical smart environment scenario we developed based on the domain analysis in Chapter 2, the network contains 138 CompModInsts. This is quite a high number, considering that each of those CompModInsts has to be processed in every action selection cycle. In an environment like ours the CompModInsts are distributed among the devices, so this is not as bad as it would be on a single processor machine. But if several of the CompModInsts run on a single device, computation can become very slow. During each cycle, a CompMod performs some basic arithmetic operations for each of its CompModInsts. Thus, to reduce computational complexity, it is beneficial to reduce the number of CompModInsts in the network.

One way to reduce the size of the network of CompModInsts is to include only those CompModInsts into the action selection process that can – directly or indirectly – contribute to the fulfillment of at least one open goal. In other words, if we view the network of CompModInsts, current percepts and goals as a directed graph, the reduced network consists of those connected components of the graph that include at least one goal (see Figure 4.8 for illustration). This can be achieved in the following way:

In the beginning, the reduced network is empty. Whenever a new goal is announced, new CompModInsts are added to the reduced network. Let us now assume that at time $t + 1$ the new goal g is announced. $R_I^{(t+1)}$ denotes the set of CompModInsts that constitute the reduced network at time $t + 1$.

The reduction of the network now takes place in two phases. In the first phase, all CompModInsts that have at least one effect which equals g are added to the reduced network:

$$R_I^{(t+1)} = R_I^{(t)} \cup \{x \mid \{g\} \cap \mathit{eff}(x) \neq \emptyset\}$$

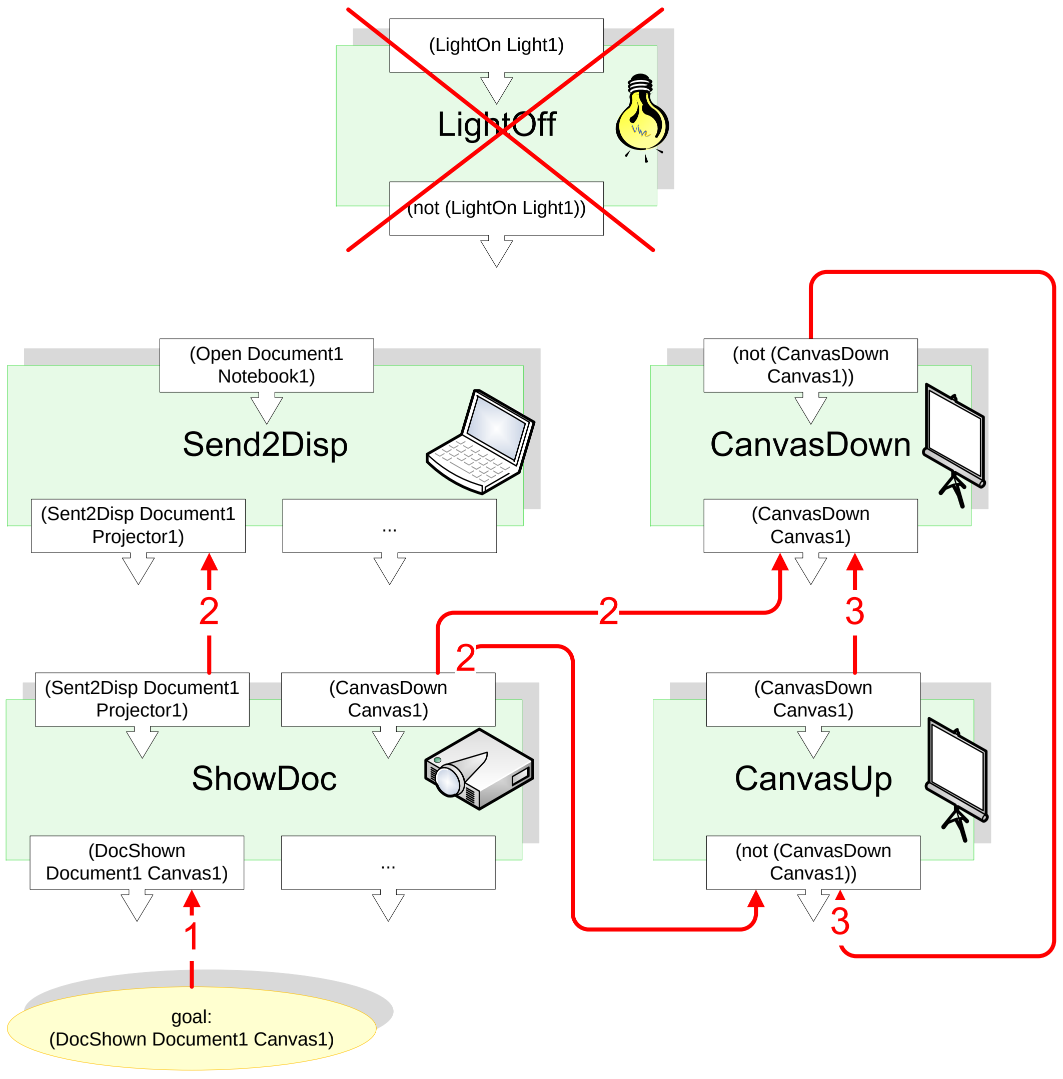

(LightOn Light1)
LightOff
(not (LightOn Light1))
(Open Document1 Notebook1)
Send2Disp
(Sent2Disp Document1 Projector1)
...
(not (CanvasDown Canvas1))
CanvasDown
(CanvasDown Canvas1)
2
2
3
2
(Sent2Disp Document1 Projector1)
(CanvasDown Canvas1)
ShowDoc
(DocShown Document1 Canvas1)
...
1
(CanvasDown Canvas1)
CanvasUp
(not (CanvasDown Canvas1))
3
goal: (DocShown Document1 Canvas1)

Figure 4.8: Building up the reduced network.

In the second phase, each CompModInst that is a predecessor or conflicter of at least one CompModInst in the reduced network is added, provided that it is not already part of the reduced network. For the sake of conciseness, we subsume the predecessors and conflicters of a CompModInst as those that come *before* the CompModInst:

$$before(x, y) \Leftrightarrow x \in Pred(y) \vee x \in Conf(y)$$

Hence, in the second phase, those CompModInsts are added to the reduced network:

$$R_I^{(t+2)} = R_I^{(t+1)} \cup \{x \mid \exists x' \in R_I^{(t+1)} \wedge \mathit{before}(x,x')\}$$

The second phase is then repeated until no more CompModInsts can be added to the reduced network. Conceptually, the reduction of the network amounts to computing the transitive closure of the *before* relation.

As an example, consider the network in Figure 4.8. The consecutive steps of the reduction are denoted by the red arrows with numbers. The reduction starts when the goal `(DocShown Document1 Canvas1)` is announced. The CompModInst *ShowDoc* has this goal as an effect. Hence, in step 1, *ShowDoc* becomes part of the reduced network. In step 2, *ShowDoc* informs its predecessors *Send2Disp* and *CanvasDown* and its conflicter *CanvasUp* that they are part of the reduced network, too. In step 3, *CanvasDown* informs its predecessor *CanvasUp*, and *CanvasUp* informs its predecessor *CanvasDown*. However, both have been informed by *ShowDoc* in step 2 and are thus already part of the reduced network. Now no CompModInst has any predecessors or conflicters that have not yet been informed, hence the reduced network is complete. Since *LightOff* is not a predecessor or conflicter of any CompModInst in the reduced network, it is not part of the reduced network. This essentially means that *LightOff* is excluded from the action selection process.

The adding of CompModInsts in the second phase is implemented via messages: Each CompModInst x' that is added to the reduced network broadcasts this fact in a message NMMsg(12: $\omega_x(a)$) (see Table 4.4). All other CompModInsts receive this message. Then each CompModInst x with $\mathit{before}(x,x')$ is added to the reduced network if it is not already part of the reduced network and broadcasts this fact, too. We say that x' is an *informer* for x: $\mathit{inf}(x',x)$. This will become important in case x' leaves the reduced network. An explanation follows in the next section.

The reduced network R_I is stored in a distributed fashion: Each CompMod only stores information about which of its own CompModInsts are part of R_I.

4.4.9 Deleting CompModInsts from the Reduced Network

CompModInsts are deleted from the reduced network in two cases:

- Whenever a goal is fulfilled, CompModInsts are deleted in two phases. In Phase 1, each CompModInst is deleted from the reduced network that could contribute to the fulfilled goal, has no other other goal in its effects and has no informers. In Phase 2, each CompModInst is deleted from the reduced network that has no more informers because they were all deleted in Phase 1, provided that it can contribute to no other goals. This is then repeated until no more CompModInsts can be deleted from the reduced network.
- Whenever a CompModInst leaves the ensemble (e.g. because the device that carries it leaves), it also leaves the reduced network. Furthermore, each CompModInst leaves the reduced network that has no other informers than the one that has left the ensemble, provided that it has no open goal in its effects. Again, this is repeated until no more CompModInsts can be deleted from the reduced network.

Consider the situation that at time $t+1$, a goal is fulfilled or a CompModInst has left the ensemble. In both cases, the reduced network is adapted in the following way:

$$R_I^{(t+1)} = R_I^{(t)} \setminus \{x \mid G^{(t+1)} \cap \mathit{eff}(x) = \emptyset \;\wedge\; \nexists x' \text{ with } \mathit{inf}(x', x)\}$$

This process is then repeated until no more CompModInsts can be deleted from the reduced network. Like adding CompModInsts, deleting them is implemented via messages. Each CompModInst x that is deleted from the reduced network broadcasts this fact in a message NMMsg(13: $\omega_x(l)$) (see Table 4.4). Each CompModInst x' with $\mathit{inf}(x, x')$ is then deleted from the reduced network if it has no other informers and no goal in its effects. It then broadcasts that it has left the reduced network, and the process continues.

4.4.10 Executability of CompModInsts

Remember that in Section 4.1.4, we defined an operator ω to be *executable* in a world W if its precondition formula holds, which is written as $\mathit{Poss}(\omega, W)$. Formally:

$$\mathit{Poss}(\omega, W) \Leftrightarrow W \models \mathit{pre}(\omega) \tag{4.38}$$

This is exactly the definition of *executable* used by the PM algorithm. However, there are two problems with this.

First, loops can occur. This has been remarked by Maes herself [Maes 1990a]. Suppose that we have two competence modules that have an equal link structure, and each destroys an effect of the other when executed. In this case a situation is possible in which both are executed alternately. This results in an "oscillating" behavior, where no other competence module has the chance to become active. To prevent this in the AdDCo algorithm, we adopt the solution suggested by Maes herself: We decrease a CompModInst's probability to become active again each time it is executed.

Second, useless actions may be executed. Consider the situation when one competence module receives a lot of activation energy due to fulfilled preconditions in every cycle. It can become active again and again, even if its effects are already fulfilled after executing it once. In the AdDCo algorithm, we therefore introduce an additional restriction: Only those CompModInsts that have at least one effect that is not already fulfilled can become active.

To account for those two changes in the AdDCo algorithm, we redefine the term *executable*. To determine an operator ω_x's executability, its competence module x is assigned a probability ε_x that changes over x's lifetime: It is set to 1.0 whenever a new goal is announced. Whenever ω_x is executed, ε_x is multiplied by λ.[14] Then, in each cycle of the action selection algorithm a random number $\kappa \in [0, 1]$ (uniformly distributed) is chosen and compared to ε_x. We say that an operator ω_x is *executable* in that cycle if:

- its preconditions hold
- not all of its effects are valid
- $\kappa \leq \varepsilon_x$

Executability is denoted by the variable $\widetilde{\varepsilon_x}$:

$$\widetilde{\varepsilon_x} = \begin{cases} \text{true} & \text{if } \omega_x \text{ is executable} \\ \text{false} & \text{otherwise.} \end{cases} \tag{4.39}$$

[14] A good value for λ that we found empirically is $1/2$.

4.4.11 The Action Selection Algorithm

Having settled the necessary preliminaries, we are now in position to present the action selection algorithm. The algorithm performs a loop. At every timestep, the following computation takes place in each CompMod. We describe it for timestep $t+1$ and CompMod c. Here, C denotes the set of CompMods in the network, X is the set of c's CompModInsts, and ε is a random number in $[0.0, 1.0]$. Furthermore, $\Xi^{(t)}$ denotes the set of CompModInsts in the network whose preconditions are fulfilled at timestep t, or, more formally, $x \in \Xi^{(t)} \Leftrightarrow Poss(\omega_x, W^{(t)})$.

Note that at some points in the algorithm, all CompMods in the ensemble have to synchronize themselves to ensure that each has correct and complete information. This is the case when the CompMods broadcast information that other CompMods need for the next step. Specifically, synchronization takes place at the end of Steps 2 and 4 in the forthcoming algorithm. It is realized in a straightforward manner: Upon completing Step 2 (or 4, respectively), each CompMod sends a message saying that it is ready to proceed to the next step. When all CompMods have sent this message, each of them knows that it is safe to proceed.

We now present the algorithm. More detailed explanations follow.

1. For each CompModInst $x \in X \cap R_I^{(t+1)}$, compute new activation due to current situation:

$$\begin{aligned}\alpha_x^{(t+1)}(w) = & \sum_{i \in pre(x) | W^{(t)} \models i} \phi \quad \textit{(activation for world state)} \\ & + \sum_{i \in eff(x) \cap G^{(t)}} \gamma \quad \textit{(activation for open goals)} \\ & - \sum_{i \in eff(x) | \neg\, i \in R^{(t)}} \delta \quad \textit{(inhibition for protected subgoals)}\end{aligned}$$

2. For each CompModInst $x \in X \cap R_I^{(t+1)}$, compute and send activation to x's predecessors, successors, and conflicters:

$$\alpha_{xz}^{(t+1)}(pred) = \sum_{i \in (pre(x) \cap eff(z)) | W^{(t)} \not\models i | x \in \bar{\Xi}^{(t)}} \alpha_x^{(t)} \quad \textit{(compute activation sent to predecessor)}$$

$$\alpha_{xz}^{(t+1)}(succ) = \sum_{i \in (pre(z) \cap eff(x)) | W^{(t)} \not\models i | x \in \Xi^{(t)}} \frac{\alpha_x^{(t)}\ \phi}{\gamma} \quad \textit{(compute activation sent to successor)}$$

$$\alpha_{xz}^{(t+1)}(conf) = \sum_{\neg\ i \in pre(x) | i \in eff(z) | W^{(t)} \models \neg\ i} \frac{\alpha_x^{(t)}\ \delta}{\gamma} \quad \textit{(compute inhibition sent to conflicter)}$$

Send ASMsg(1: $\omega_z(r)$, 2: $\alpha_{xz}^{(t+1)}(pred)$, 3: $\alpha_{xz}^{(t+1)}(succ)$, 4: $\alpha_{xz}^{(t+1)}(conf)$) and synchronize

3. For each CompModInst $x \in X \cap R_I^{(t+1)}$, compute new activation sent to x by successors, predecessors, and conflicters via messages:

$$\begin{aligned}
\alpha_x^{(t+1)}(s) = & \sum_{x \in Pred(z) \cap \bar{\Xi}^{(t)}} \alpha_{zx}^{(t+1)}(pred) \quad \textit{(activation by successors)} \\
+ & \sum_{x \in Succ(z) \cap \Xi^{(t)}} \alpha_{zx}^{(t+1)}(succ) \quad \textit{(activation by predecessors)} \\
- & \sum_{x \in Conf(z)} \alpha_{zx}^{(t+1)}(conf) \quad \textit{(inhibition by conflicters)}
\end{aligned}$$

4. For each CompModInst $x \in X \cap R_I^{(t+1)}$, compute ω_x's executability $\widetilde{\varepsilon_x}$ according to (4.39) as well as preliminary activation level $\alpha_x^{(t+1)}(a)$ and send them in a message:

$$\alpha_x^{(t+1)}(a) = \alpha_x^{(t)} + \alpha_x^{(t+1)}(w) + \alpha_x^{(t+1)}(s)$$

Send ASMsg(5: $\omega_x(s)$, 6: $\alpha_x^{(t+1)}(a)$, 7: $\widetilde{\varepsilon_x}$) and synchronize

5. Compute overall activation level in the network:

$$\alpha^{(t+1)} = \sum_{y \in R_I^{(t+1)}} \alpha_y^{(t+1)}(a)$$

6. For each CompModInst $x \in X \cap R_I^{(t+1)}$, normalize activation level to keep total activation $\pi|R_I^{(t+1)}|$ constant:

$$\alpha_x^{(t+1)} = \frac{\alpha_x^{(t+1)}(a)\, \pi\, |R_I^{(t+1)}|}{\alpha^{(t+1)}}$$

7. Compute set of execution candidates:

$$E^{(t+1)} = \{\, x \in R_I^{(t+1)} \mid \alpha_x^{(t+1)} = \max_{z \in R_I^{(t+1)}, \widetilde{\varepsilon_z} = \text{true}} \alpha_z^{(t+1)} \wedge \alpha_x^{(t+1)} > \theta \,\}$$

8.a If $E^{(t+1)} = \{x_0\}$, so we have a unique execution candidate: Execute operator:

$$\begin{aligned}
W^{(t+1)} &= Do(\omega_{x_0}, W^{(t)}) \quad \textit{(new world state)} \\
G^{(t+1)} &= (G^{(t)} \setminus \{i \mid W^{(t+1)} \models i\}) \cup (R^{(t)} \cap \{i \mid W^{(t+1)} \not\models i\}) \quad \textit{(open goals)} \\
R^{(t+1)} &= (R^{(t)} \setminus \{i \mid W^{(t+1)} \not\models i\}) \cup (G^{(t)} \cap \{i \mid W^{(t+1)} \models i\}) \quad \textit{(protected subgoals)} \\
\theta &= \theta_0 \quad \textit{(reset activation threshold)}
\end{aligned}$$

If $x_0 \in X$

$$\varepsilon_{x_0} = \varepsilon_{x_0} * \lambda \quad \textit{(decrease x's probability for becoming executable)}$$

8.b Else

$$\theta = \theta * 0.9$$

For each $x \in E^{(t+1)} \cap X$:

$$\alpha_x^{(t+1)} = \alpha_x^{(t+1)} + \varepsilon \quad \textit{(break activation level ties)}$$

Figure 4.9 depicts which variables are involved in the calculation of the activation levels $\alpha_x^{(t+1)}$ and $\alpha_z^{(t+1)}$ of two CompMods x and z at time $t+1$. To preserve readability, just the results of the calculations at time t, namely $\alpha_x^{(t)}$ and $\alpha_z^{(t)}$ are depicted, not the calculations themselves. Note that this figure is only correct if x and

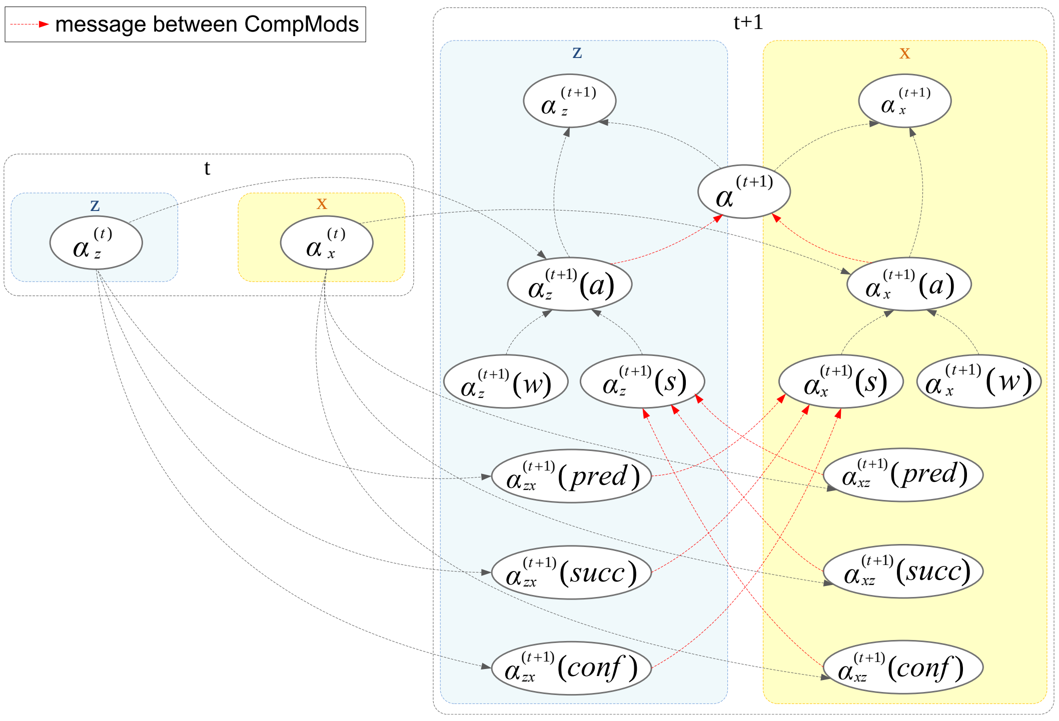


Figure 4.9: Calculation of the activation level of CompMods x and z at time $t + 1$. The information flow depicted as dotted red lines is realized via messages between CompMods.

y are the only CompModInsts in the reduced network. If there are more CompModInsts, more nodes and arrows must be included. As described above, some parts of the calculation require inter-CompMod communication, which is realized via messages. In Figure 4.9, this is depicted as red dotted lines. Furthermore, the calculation of the overall activation level $\alpha^{(t+1)}$ deserves a more detailed explanation. All CompModInsts broadcast their preliminary activation level $\alpha^{(t+1)}(a)$, which is depicted as the red dotted lines leading towards $\alpha^{(t+1)}$. Each CompMod receives those messages from all other CompMods. In the following, each CompMod computes $\alpha^{(t+1)}$ individually.

4.4.12 Differences to the PM Algorithm

Although the action selection part of the AdDCo algorithm is similar to the PM algorithm, there are important differences between the two algorithms. In this section, we describe these differences. We start with a short recapitulation of differences introduced earlier in this chapter. Afterwards, some differences not yet mentioned are explained in detail.

As said before, the PM algorithm and the AdDCo algorithm differ in the following aspects:

- Whereas the PM algorithm runs on a single machine, the AdDCo algorithm is designed to be distributed across the devices in the ensemble and can adapt to changing ensemble structures. This makes it applicable in dynamic environments.
- The AdCo algorithm is more goal-oriented than the PM algorithm: Only those operators take part in action selection that can contribute to the fulfillment of at least one open goal. This results in more predictable behavior and shorter action sequences, which is shown in the next section.
- To be executable, an operator's preconditions must be fulfilled in the PM algorithm. In the AdDCo algorithm, additional conditions must be met: Not all effects must be fulfilled before execution, and each operator can only be executed with a certain probability that falls with every execution of the operator. This is to prevent oscillating behavior.

There are two more important differences between the two algorithms:

- Both the PM algorithm and the AdDCo algorithm allow at most one action to be executed per cycle. The reason for this is simple. Consider the case that two actions A and B have the same activation level, and that their activation level is above the activation threshold. Let us assume further that A has literal x as an effect, and B has literal $y = \neg x$ as an effect. If we allowed both to be executed, the world state would become inconsistent, because obviously a literal and its complement cannot hold at the same time. Thus, both in the PM algorithm and in the AdDCo algorithm no action is executed if there is more than one candidate, all of which have the same activation level. The question that arises here is how to further proceed in the next cycle. The PM algorithm proceeds as normal. However, this may cause problems: If all the candidates have the same link structure (which is not unlikely if they have the same activation level), they receive an equal amount of energy in the next cycle, and so on. In other words, the same tie reoccurs again and again, and as long as there is no other action with more activation energy, no action can be executed. To overcome this problem, the AdDCo algorithm breaks ties arbitrarily: We add a random number in $[0.0, 1.0]$ to each candidate's activation level. This ensures that they have different activation levels in the next cycle, so that there is only one action with the highest activation level. At the same time, the random number is sufficiently small to give another action the chance of having the highest activation level and to be executed. Another solution would be to introduce an additional negotiation phase at the end of each cycle. Here, ties could be broken if they occur. However, the benefit of avoiding an extra cycle in the rare cases where ties occur does not justify the overhead introduced by a negotiation phase.

- The number of preconditions and effects a module has affects the amount of energy it receives. In Maes' opinion, this favors modules with many preconditions/effects over modules with few. Therefore, in the PM algorithm the energy an action receives is divided by the number of its preconditions/effects. Furthermore, Maes wants modules that have the same preconditions to compete with one another. The same holds for modules that have the same effects. Hence, in the PM algorithm the amount of energy a module receives for a precondition/effect is divided by the number of modules that share this precondition/effect. After careful consideration and experimentation, we chose not to perform any of those divisions in the AdDCo algorithm. The reason is that we, like Toby Tyrrell before us [Tyrrell 1993], found that those divisions do not improve the network's action selection behavior. Omitting them saves us computational effort without impairing the solution quality.

4.5 Evaluation of the AdDCo Algorithm

Having presented the AdDCo algorithm in detail, we now evaluate its performance empirically with the help of four scenarios from the domain of smart meeting rooms.[15] All four scenarios consist of the following steps:

1. All CompMods enter the ensemble, introduce themselves and build up their part of the network of CompModInsts.
2. All literals describing the current state of the world and the goals to be fulfilled are provided to the CompMods by the intention analysis. Literals that

[15]In [Maes 1990b], Maes evaluates the performance of her algorithm with the help of some simple toy problems. We chose not to use them for the evaluation in this chapter as they offer little insight into how the algorithms perform in real-world settings.

are not mentioned are assumed to be false (closed world assumption).

3. The CompMods build up the reduced network according to the goals issued in step 2.
4. The CompMods perform the action selection process until all goals have been fulfilled.

For comparison, we have reimplemented the PM algorithm. We compare the performance of the PM algorithm to the performance of the AdDCo algorithm. It is important to remember that, in contrast to the PM algorithm, the AdDCo algorithm can handle dynamically changing ensemble structures. In every scenario, the AdDCo algorithm generates CompModInsts and links dynamically according to the devices and CompMods present in the ensemble, whereas the PM algorithm requires that the competence modules are "hand-wired". Thus, this benchmark is not fully realistic as the PM algorithm cannot be applied in smart ad-hoc environments. It is therefore run as a simulation. For reasons of simplicity, the AdDCo algorithm, too, runs on one machine, but the CompMods are not "aware" of this fact. The AdDCo algorithm is exactly the way it would be if the CompMods were physically distributed among several devices, i.e. the CompMods communicate over the network via multicast. However, for this benchmark the CompMods do not send action commands to the ECo services, i.e. they do not drive any real devices as this is not required for the benchmark.

For all four scenarios, we compare:

- the results of both algorithms in terms of the number of competence modules/CompMods required to model the scenario
- the number of cycles each algorithm takes during action selection (step 4 in the above list)
- the length (= number of actions) of the solution found[16]

For the AdDCo algorithm, we furthermore analyze:

- the number of CompModInsts in the full and the reduced network
- the number of messages sent by the CompMods for network maintenance (i.e. messages concerning the world state) and action selection[17]

[16]Note that the competence modules in the PM algorithm do not perform step 1 and step 3 as the network of competence modules is fixed at design time.

[17]Note that in the PM algorithm, there are no CompModInsts and the modules do not communicate via a network, hence we cannot analyze these parameters for the PM algorithm.

At first, we describe the four different scenarios in detail. After that, we present the results of the runs of both algorithms with the scenarios.

Note that a formal complexity analysis of both algorithms is a non-trivial task and remains an open research question. We pick up on this issue in Section 8.2.

4.5.1 Scenario 1: Adjusting the Light Level

The ensemble in this scenario consists of 14 devices: six lights and eight canvasses, two of which are in front of the windows so they also act as sunblinds (Canvas7 and Canvas8). The initial world state, the user goals and an optimal action sequence are displayed in Table 4.5.

Table 4.5: Scenario 1

initial state	–
goal state	Light1 on Light2 on Canvas7 down
optimal action sequence	turn on Light1 turn on Light2 lower Canvas7

The initial world state is empty. Recall that we make use of the closed world assumption: Any literal not mentioned is assumed to be false. In this scenario, this means that all lights are switched off and all canvasses are up. The scenario resembles a situation where a speaker walks into a room and finds that the light conditions are not appropriate for her/his upcoming talk: The sun is too bright; therefore Canvas7 must be lowered. Furthermore, in the back of the room where the audience is sitting, the ambient light is too low to take notes during the talk. Thus, Light1 and Light2 should be turned on.

4.5.2 Scenario 2: Projector Scenario I

In this scenario (see Table 4.6), we have 16 devices: eight canvasses, a notebook, two documents, four fixed projectors (Projector1 to Projector4) and a movable projector (Projector5) which is mounted on a moving head and can show a document on any of the eight canvasses. This scenario resembles a situation in which

Table 4.6: Scenario 2

initial state	Document1 opened on Notebook1
	Document2 opened on Notebook1
goal state	Document1 shown on Canvas3
	Document2 shown on Canvas1
optimal	Notebook1 sends Document1 to Projector3
action	Notebook1 sends Document2 to Projector1
sequence	Canvas3 is lowered
	Canvas1 is lowered
	Projector1 shows Document1 on Canvas3
	Projector3 shows Document2 on Canvas1

a speaker wants to show two different presentations at a time because each contains a figure s/he wants to compare to the figure in the other presentation. In this scenario, a high-level service is available on each projector that can fetch the presentation from the speaker's notebook, open and display it. Because one cannot rely on such services being available in every environment, Scenario 4 is a more complex and fine-grained version of this scenario. Scenario 4 contains a video crossbar instead of the high-level services in this scenario.

Table 4.7: Scenario 3

initial state	Camera1 points to Canvas1
	speaker is in front of Canvas3
goal state	speaker tracked
optimal	switch on Camera1
action	Camera1 turns to Canvas3
sequence	Camera1 tracks speaker
	after 20 seconds: speaker moves to Canvas2
	Camera1 turns to Canvas2
	Camera1 tracks speaker
	after 40 seconds: speaker moves to Canvas4
	Camera1 turns to Canvas4
	Camera1 tracks speaker

4.5.3 Scenario 3: Tracking the Speaker with a Movable Camera

This scenario comprises nine devices: eight canvasses and a movable camera which can capture any of the eight canvasses. This is a scenario with changing sensor data: In the beginning, the speaker stands in front of Canvas3. After 20 seconds, s/he moves towards Canvas2 and after 40 seconds s/he moves to Canvas4 (see Table 4.7). This resembles a situation where a speaker walks around a meeting room equipped with several canvasses, briefly stopping in front of every canvas and explaining what is depicted on this canvas. The goal is to track the speaker with the camera while s/he is moving from canvas to canvas.

Table 4.8: Scenario 4

initial state	Notebook1 hosts Document1
	Notebook2 hosts Document2
goal state	Document1 shown on Canvas3
	Document2 shown on Canvas1
optimal action sequence	Notebook1 maximizes Document1
	Notebook2 maximizes Document2
	Canvas3 is lowered
	Canvas1 is lowered
	Crossbar connects Notebook1 to Projector3
	Crossbar connects Notebook2 to Projector1
	Projector3 shows Document1 on Canvas3
	Projector1 shows Document2 on Canvas1

4.5.4 Scenario 4: Projector Scenario II

This scenario was our introducing example in Section 4.3. In fact, it is a more complex version of Scenario 2. We have two notebooks instead of one, each hosting a document that is to be shown on a dedicated canvas. Furthermore, there is a video crossbar that has the notebooks as inputs and the projectors as outputs. This crossbar can connect any of its inputs to any output. Overall, this scenario (see Table 4.8) comprises 18 devices: eight canvasses, two notebooks, two documents, a video crossbar, four fixed projectors (Projector1 to Projector4) and a movable projector (Projector5). The goal is to show both documents on dedicated canvasses.

4.5.5 Results

Table 4.9 shows the results of the runs of the four scenarios for the PM and the AdDCo algorithm. For the PM algorithm, we just did one run for every scenario as the results do not vary from run to run. For the AdDCo algorithm, we did 20 test runs for every scenario. This is due to the fact that the results can vary slightly from

Table 4.9: Results

	Scenario 1	
algorithm	PM	AdDCo
parameters	$\pi = 20$	$\pi = 5$
	$\theta = 45$	$\theta = 45$
	$\phi = 20$	$\phi = 20$
	$\gamma = 50$	$\gamma = 50$
	$\delta = 40$	$\delta = 40$
number of competence modules/CompMods	28	28
number of CompModInsts in full/reduced network	–/–	28/6
number of cycles	3	10
length of action sequence	3	3
	Scenario 2	
algorithm	PM	AdDCo
parameters	$\pi = 3$	$\pi = 5$
	$\theta = 45$	$\theta = 45$
	$\phi = 20$	$\phi = 20$
	$\gamma = 50$	$\gamma = 50$
	$\delta = 40$	$\delta = 40$
number of competence modules/CompMods	212	23
number of CompModInsts in full/reduced network	–/–	58/12
number of cycles	7	20/23
length of action sequence	6	6/7
	Scenario 3	
algorithm	PM	AdDCo
parameters	$\pi = 15$	$\pi = 5$
	$\theta = 45$	$\theta = 45$
	$\phi = 1$	$\phi = 20$
	$\gamma = 300$	$\gamma = 50$
	$\delta = 0$	$\delta = 40$
number of competence modules/CompMods	74	4
number of CompModInsts in full/reduced network	–/–	18/18
number of cycles	∞	64/65
length of action sequence	∞	7
	Scenario 4	
algorithm	PM	AdDCo
parameters	$\pi = 80$	$\pi = 5$
	$\theta = 45$	$\theta = 45$
	$\phi = 20$	$\phi = 20$
	$\gamma = 50$	$\gamma = 50$
	$\delta = 40$	$\delta = 40$
number of competence modules/CompMods	154	32
number of CompModInsts in full/reduced network	–/–	102/52
number of cycles	22	35
length of action sequence	22	9

run to run: As the algorithm breaks ties arbitrarily if two CompModInsts have the same activation level, the number of cycles and/or the action sequence generated varies in Scenarios 2 and 3.

Maes remarked that finding a good parameter setting for the PM algorithm is nontrivial. This is a drawback of the algorithm. Maes herself tuned the parameters by hand in a trial-and-error fashion [Maes 1989]. This is what we have done here for the PM algorithm, too. We found that a different parameter setting is required for the PM algorithm for each scenario (see Table 4.9). This is a problem in ad-hoc environments, where neither the time nor the means for hand-tuning the parameters are available. For the AdDCo algorithm, we have also tried a variety of parameter settings. As a result, we found a parameter setting that works well with the AdDCo algorithm for all scenarios we evaluated. Thus, it seems to be quite robust. Whether the same parameters are suitable for all possible scenarios is, however, an open question. We did not answer this question because our focus lies on the question whether it is *in principle* possible to use a decentralized algorithm for controlling smart environments.

4.5.6 Discussion

We now relate the results of the PM algorithm to those of the AdDCo algorithm with the help of Table 4.9. The PM algorithm performs slightly better in terms of the number of cycles for Scenarios 1 and 2 and even in terms of the length of the generated action sequence for Scenario 2. Here the AdDCo algorithm does in some cases not generate an optimal action sequence. In some runs, it includes one useless action: The document is sent to the movable projector (Projector5) first, then the system ”realizes“ that using the fixed projectors is easier as they point to the correct canvasses and thus need not be turned.

In Scenarios 3 and 4, the AdDCo algorithm outperforms the PM algorithm. In Scenario 3, the PM algorithm does not terminate but the camera keeps turning around. In contrast, the AdDCo algorithm finds the optimal action sequence and manages to track the speaker wherever s/he goes. In Scenario 4, the PM algorithm performs better in terms of the number of cycles, but the action sequence generated contains 22 actions, where 8 is the optimum. The AdDCo algorithm takes 33 cycles, but generates an action sequence consisting of 9 actions, which is almost optimal.

Another issue is the number of CompMods needed to model a scenario. Consider that preconditions and effects in the PM algorithm are conjunctions of negated and non-negated symbols. Hence, variables are not allowed. This means that each combination of devices that can occur in the preconditions and effects of an action description must be hard-coded in a different competence module. This typically results in a high number of competence modules. However, if variables were the only difference between the two algorithms, the number of competence modules in the PM algorithm would equal the number of CompModInsts in the AdDCo algorithm because the instantiation process in the AdDCo algorithm would lead to the same number of operators. For Scenario 1, this is the case: The number of competence modules in the PM algorithm equals the number of CompMods/CompModInsts in the AdDCo algorithm. For Scenarios 2, 3, and 4, however, the number of competence modules in the PM algorithm is considerably higher than the number of CompMods and also higher than the number of CompModInsts in the AdDCo algorithm. This is due to the fact that these scenarios include persistent actions, and that the two algorithms have different means of dealing with persistent actions. The details are explained in Section 5.2. For now, it is sufficient to note that in the AdDCo algorithm, the modelling effort for persistent actions is lower than in the PM algorithm, which also leads to a smaller number of operators.

Now consider that in the AdDCo algorithm only the operators/CompModInsts in the reduced network actually take part in the action selection process. Usually, their number is even much smaller: In Scenarios 1, 2, and 4 it is only a fraction of the number of CompModInsts in the full network. Hence, both the effort needed to model a scenario and the computation effort per action selection cycle are typically considerably lower for the AdDCo algorithm than for the PM algorithm.

One can also analyze the network traffic caused by the AdDCo algorithm in the different scenarios. We depict the messages sent in the initial phase (Figure 4.10), the average number of messages sent during one cycle of the action selection process (Figure 4.11), and the total number of messages sent during action selection (Figure 4.12) in 20 runs of the AdDCo algorithm in Box-Whisker plots. The box represents the interquartile range, the horizontal line in the box represents the median, and the ends of the whiskers represent the minimum/maximum. By “initial phase” we mean the beginning of each scenario, when the CompMods enter the ensemble and build up the network.

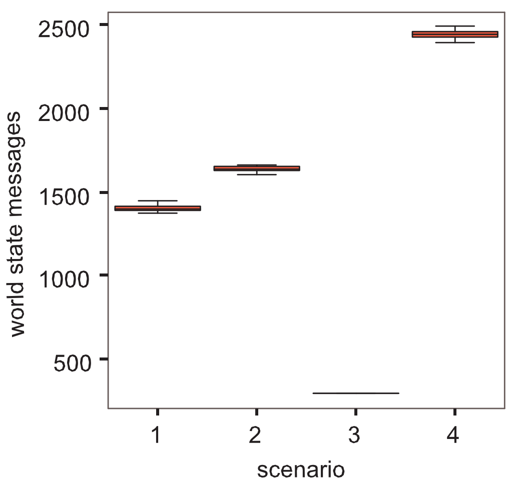


Figure 4.10: The total number of messages sent during the initial phase.

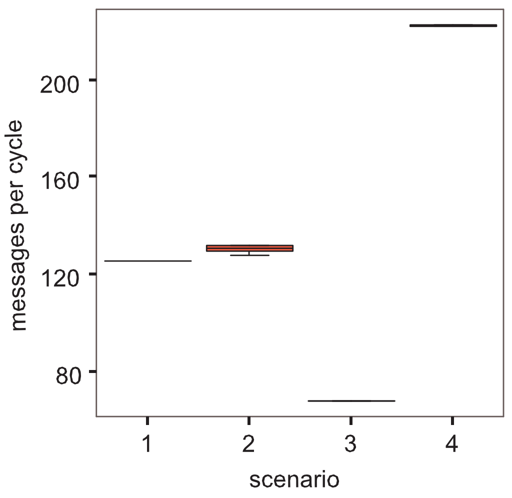


Figure 4.11: The average number of messages per action selection cycle.

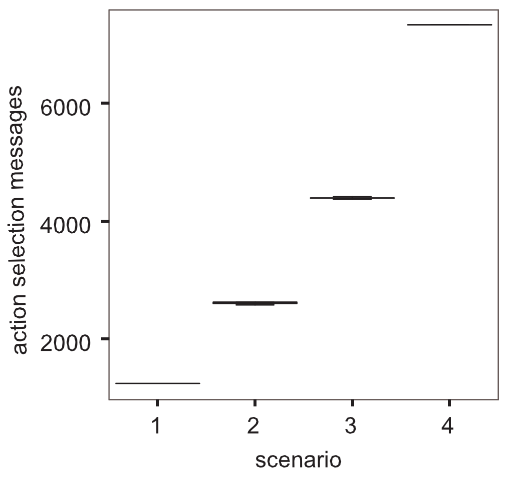


Figure 4.12: The total number of messages sent during action selection.

Variation from run to run in the number of messages is usually due to the randomness introduced by the network connection. For example, if by chance many CompMods enter the ensemble at the same time, they all send out the message *NMMsg(6: σ_y)* at once. The CompMods that are already present in the ensemble each answer by sending the message *NMMsg(1: W, 3: G, 5: R, 7: σ_y, 10: τ)*. In this case, the number of messages sent is smaller than if all CompMods enter the ensemble one after the other. The reason is that in this case with every new CompMod the number of messages of the form *NMMsg(1: W, 3: G, 5: R, 7: σ_y, 10: τ)* increases.

One thing is obvious: The more CompMods and CompModInsts in a scenario, the more messages are sent in the initial phase and the higher the average number of messages sent during each cycle of the action selection mechanism (see Figures 4.10 and 4.11). This correlation can explained with the “bookkeeping costs” of generating and maintaining the world model as well as distributed synchronization: It rises with each single CompMod and with each single CompModInst. This leads us to the conclusion that, in order to reduce network traffic, it is beneficial to model domains with as few CompMods/CompModInsts as possible. On the other hand, the total number of messages sent during action selection depends on the length of the action sequence generated as well as the number of CompMods/CompModInsts (see Figure 4.12). The length of action sequences is hard to anticipate for the authors of action descriptions, especially in distributed settings where action descriptions are written by different authors. However, to reduce the total number of messages sent, once again it is beneficial to model domains with as few CompMods/CompModInsts as possible.

4.6 Classification of the AdDCo architecture

In the following, we discuss how the AdDCo architecture can be classified with respect to two models. The architecture can be seen from different viewpoints. Here, we consider it on the single-agent level and on the multi-agent level.

4.6.1 On the Single-Agent Level: A Temporally Layered Architecture

On the single-agent level it can be seen as a new kind of hybrid architecture. Recall that in Section 3.3 we introduced two kinds of hybrid architectures: vertically layered and horizontally layered. Like the approaches discussed there, the AdDCo architecture consists of separate subsystems that can be viewed as layers: The reduction of the network according to the current open goals (described in Section 4.4.8) can be viewed as a deliberative subsystem, while action selection (described in Section 4.4.11) can be seen as a reactive subsystem. Yet the AdDCo architecture is neither vertically nor horizontally layered. In both vertically and horizontally layered architectures, the different subsystems produce some output (actions to be executed by the actuators). There is either a component that arbitrates between the outputs of the different subsystems (as in Ferguson's TouringMachines [Ferguson 1992]) or the output of one layer is fed into another layer, which splits it into smaller, executable parts (as in the AuRA architecture [Arkin & Mackenzie 1994]). In the AdDCo architecture, however, only the reactive subsystem (i.e. action selection) actually produces output. The deliberative subsystem performs a preprocessing step for the reactive subsystem. It changes the world model according to the current goals by pruning operators that are not relevant in the current context. It thus eases the reactive subsystem's work. One could also interpret the deliberative layer as a compilation layer: Upon the disclosure of every new goal, the deliberative subsystem compiles a custom-made world model for the reactive subsystem to work with. Thus, as the different layers correspond to consecutive phases in the algorithm, we call this type of hybrid architecture a *temporally layered architecture* (see Figure 4.13).

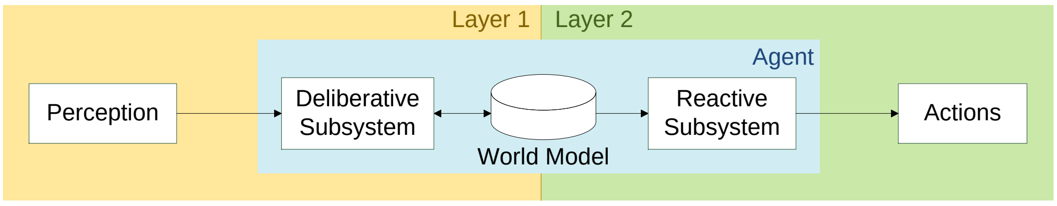


Figure 4.13: The AdDCo algorithm: a temporally layered architecture.

4.6.2 On the Multi-Agent-Level: Different Phases in Wooldridge's and Jennings' CDPS model

On the multi-agent level we can classify the AdDCo architecture in terms of Wooldridge's and Jennings' four-stage Cooperative Distributed Problem Solving (CDPS) model [Wooldridge 2001]. This is illustrated in Figure 4.14.

Wooldridge's and Jennings' CDPS model consists of the following four stages:

1. Recognition: During this stage, an agent has a goal and realizes that it is beneficial to ask other agents to cooperate for achieving the goal. A possible reason is that the agent is not able to fulfill the goal alone, e.g. because this requires information that another agent has. Another reason is that the agent could fulfill the goal alone, but is not willing to, for example because a better solution can be found by cooperating with another agent. In the AdDCo algorithm, the recognition phase corresponds to the phase where a goal is issued to all CompMods. The CompMods add this new goal to their world model. Those CompMods that have CompModInsts that can fulfill the goal (i.e. that have the goal as one of their effects) realize that they are part of the reduced network. The members of the reduced network will communicate in

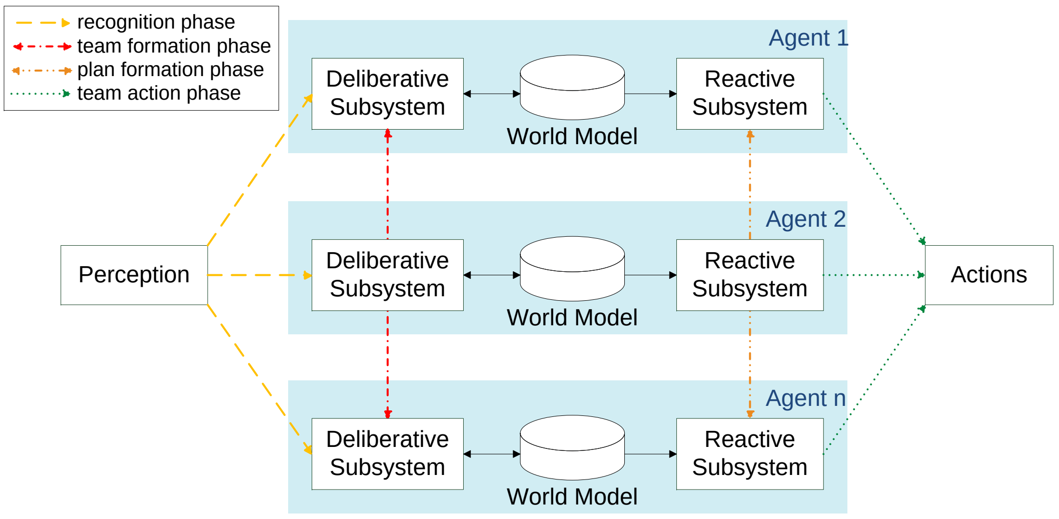


Figure 4.14: The AdDCo algorithm in terms of Wooldridge's CDPS model [Wooldridge 2001].

the next phases in order to fulfill the goal. The CompMods always communicate to find a solution – even if an individual CompModInst could fulfill the goal alone. The problem is that there might be other CompModInsts that could also fulfill the goal, but at most one at a time must be executed. Hence, it has to be decided which of them will be executed.

2. Team formation: In this stage, the agent seeks other agents to cooperate in order to fulfill the goal. Upon success, there will be a group of agents that has some kind of nominal commitment to act cooperatively. Wooldridge and Jennings point out that at the end of this stage, this group of agents will agree on the *ends* they want to achieve (i.e. to act cooperatively). However, they do not agree on the *means* (i.e. which actions should be taken by the group to fulfill the goal) at this stage. It is important to note that the agents will not form a team if they do not believe that they can achieve the goal. In the AdDCo algorithm, the team formation phase corresponds to the deliberative phase, i.e. the process of network reduction. All CompMods that have CompModInsts which can fulfill the goal, i.e. those that have joined the reduced network in the recognition phase, tell their predecessors and conflicters that they are part of the network, too. These in turn inform their predecessors and conflicters and so on. In the end, the reduced network contains exactly those CompModInsts that can be part of an action sequence that fulfills the

goal. This network can be regarded as a team of agents. The agents' only forming a team if they think they can fulfill the goal is achieved implicitly: If no CompModInst has the goal as its effect, then no CompModInst can start building up the network by informing its predecessors and conflicters. Thus, the network will contain 0 CompModInsts.

3. Plan formation: In this stage, the agents negotiate and argue which course of action to take to reach the desired goal. They might have to decide which sequence of actions should be performed if different agents know of different alternatives. Some agents might have objections to some actions, etc. Thus, the group of agents must reach an agreement in this stage. In the AdDCo algorithm, this phase corresponds to the reactive phase, i.e. the action selection process. The spreading activation algorithm is carried out in order to select actions that lead towards the goal. Every CompModInst receives energy from other CompModInsts which view it as beneficial and loses energy due to CompModInsts that view it as disadvantageous. At every point in time, the activation level of a CompModInst can be interpreted as the group's collective degree of belief in the ability of this CompModInst to bring the group closer to the goal.

4. Team action: In this stage, the plan the agents have agreed on is executed. In the AdDCo algorithm, this stage corresponds to a CompModInst "winning" a cycle of the action selection algorithm and its action being executed. This means that it sends a command to an actuator (i.e. the ECo service that drives the respective device) for execution. Notice that in the AdDCo algorithm, the plan formation phase and the team action phase are interleaved: Once a plan step has been chosen, it is executed right away. Afterwards, the changed world state is taken into account to select the next plan step and so on.

4.7 Chapter Summary

In this chapter, we have described the AdDCo algorithm, an algorithm for device cooperation that is based on Maes' action selection mechanism. It allows for *goal-based interaction*, which we introduced in Section 3.1.2. Furthermore, the algorithm as we have described it until now fulfills the following requirements introduced in Section 2.2:

- **Spontaneity**: Our approach does not rely on learning: Devices form an ad-hoc ensemble at run-time. They build up a network of CompModInsts which is able to perform action selection right away, without a training phase.
- **Action sequences**: The AdDCo algorithm is an action selection algorithm and is able to generate action sequences of moderate length. As emphasized in Section 2.2, this is sufficient for typical smart environment scenarios.
- **Robustness**: If a device leaves, its CompMods and CompModInsts leave, too. Should this happen unexpectedly, the rest of the ensemble notices this after a few seconds because the CompMods of the leaving device do not send any synchronization messages anymore. They and their CompModInsts are then excluded from the network. The rest of the ensemble still works as before. If there are other CompModInsts that have the same effects as the CompModInsts that have left, the functionality of the network is not even hampered. Note that we did not include this detection mechanism into our description of the algorithm in this chapter for reasons of simplicity. We believe that it is sufficiently straightforward to be reproducible.

However, the following requirements are only partially fulfilled by the algorithm yet:

- **Rationality**: The algorithm is based on small software components (the CompMods), each of which has only partial knowledge of the world. This sometimes results in suboptimal behavior of the algorithm. We have seen this in Section 4.5: Unnecessary actions can be chosen and executed. Thus, the requirement *rationality* is only partially fulfilled. This problem can not be solved completely. The question is whether users accept a system based on the AdDCo algorithm although it sometimes shows non-optimal behavior. We will try to answer this question with the help of a user study in Chapter 6.
- **Flexibility**: The AdDCo algorithm as we have described it until now supports *flexibility* to a certain extent: The network of devices is built up at run-time, and action sequences are generated at run-time, too. However, one problem remains: Until now, preconditions and effects in operator schemes must be function-free first order literals. Quantifiers are not allowed. In smart environments, however, authors of action descriptions often need to express that some effect holds for all devices of a certain type except one

single dedicated device. Without quantifiers, this requires that authors know which devices of a certain type will be part of the ensemble at the time of writing. This contrasts with the requirement *flexibility*. In the next chapter, we discuss this problem in detail and show how effects can be extended to contain universally quantified literals.

- **Distributedness**: Recall that we divided the requirement *distributedness* into *run-time modularization* and *design-time modularization*. As there is no central controlling component in the AdDCo algorithm, it perfectly supports *run-time modularization*. Furthermore, as authors of action descriptions only need to know about the actions they want to describe (and possibly precedent and antecedent actions), device manufacturers can develop their devices and the descriptions of the device actions with little information about other vendors' devices. Thus, *design-time modularization* is already supported by the AdDCo algorithm. However, it can be enhanced even further by hiding CompMods and CompModInsts inside the devices. In the next chapter, we show how this can be achieved.

Furthermore, the AdDCo algorithm does not fulfill the requirement *support for persistent actions* yet. In the next chapter, we therefore introduce another enhancement to the algorithm which adds support for persistent actions.

CHAPTER 5

Enhancing the AdDCo Algorithm

Contents

In the previous chapter, we have discussed how the AdDCo algorithm fulfills the requirements identified in Section 2.2. We have come to the conclusion that it could be improved by enhancing flexibility, supporting persistent actions and enhancing design-time modularization. In this chapter, we therefore introduce three additional extensions to the AdDCo algorithm: In Section 5.1, we argue why it is useful to support universally quantified effects: It improves flexibility. We then describe how we implemented this in the AdDCo algorithm. In Section 5.2, we describe the persistent action problem in detail. We then discuss how this problem can be solved in ad-hoc environments using planning and how it can be solved more elegantly using the AdDCo algorithm. In Section 5.3, we propose to hide CompMods and CompModInsts inside the devices to improve design-time modularization. In contrast to the other two extensions, this enhancement has not actually been realized in our implementation. Therefore we point out how the AdDCo algorithm must be altered if one decides to implement this extension.

5.1 Enhancing Flexibility: Supporting Universally Quantified Effects

Until now, we have assumed that preconditions and effects are function-free first order literals with no quantifiers. However, in smart environments, situations occur where we need to express that an effect holds for a certain set of devices. Let us look at an example: We have an ensemble with seven documents (*Doc1 – Doc7*) and one notebook (*NB1*). The notebook carries a CompMod called *Maximize* that can open and maximize documents on the notebook screen. Consider that the notebook screen is an exclusive resource. In other words, only one document can be maximized at a time. This must be reflected in the operator description. Here is the *Maximize* operator for maximizing *Doc1* in PDDL:

```
(:action Maximize
  :parameters ()
  :precondition (Hosts Doc1 NB1)
  :effect (and (isMax Doc1 NB1)
               (not (isMax Doc2 NB1))(not (isMax Doc3 NB1))
               (not (isMax Doc4 NB1))(not (isMax Doc5 NB1))
               (not (isMax Doc6 NB1))(not (isMax Doc7 NB1)))
```

Notice that we need seven literals as effects – one for each document. First of all, we have to express that *Doc1* is maximized after the action has been executed. Therefore, we need the literal `(isMax Doc1 NB1)`. The other six literals may seem redundant at a first glance because – as explained in the last chapter – we make use of the closed world assumption. Hence, any literal that is not mentioned is assumed to be false. Why do we need the six extra literals then? In fact, the closed world assumption can only be used for the *current world state* because it describes how the world *is*. In contrast, an *action description* like our example *Maximize* describes how the world *changes* when the action is executed. Hence, we have to include the six literals because the action can change each literal's truth value.

This can best be described via example: Consider the situation that before executing the *Maximize* action, *Doc2* is maximized on *NB1*. In this case, the world state at this time would include the literal `(isMax Doc2 NB1)`. If we did not include the effect `(not (isMax Doc2 NB1))` in the action description, the world state after the execution of *Maximize* would include both the fact that *Doc1* is maximized *and* the fact that *Doc2* is maximized on the same notebook screen. Obviously, this would be inconsistent with the laws of the real world. The same holds for all other documents in the ensemble, in the above case *Doc3*, *Doc4*, *Doc5*, *Doc6*, and *Doc7*. Hence, without universal quantification, we have to include one effect per document.

Obviously, this is a large modelling overhead. Yet there is another problem: The action description above is only correct in environments with seven documents. Hence, in order to write such an action description, one must know at design-time how many documents will be in the ensemble at run-time. In dynamic ensembles where devices can join and leave anytime, such prior knowledge is not available. Hence, it is not possible to model the *Maximize* action for dynamic environments without universal quantification. In other words: This kind of modelling does not fulfill the requirement *flexibility* we introduced in Section 2.2.

It would be beneficial if we could express things like “All devices of a certain type other than device *x*”. This requires that our effects support universal quantification. Here is the respective operator in PDDL:

```
(:action Maximize
  :parameters ()
  :precondition (Hosts Doc1 NB1)
  :effect (and (isMax Doc1 NB1)
               (forall (?x - Document)
                   (when (not (= ?x Doc1))
                         (not (isMax ?x NB1)))))     (5.1)
```

Now the effects can be read as “*Doc1* is maximized on *NB1*. All documents that do not equal *Doc1* are not maximized on *NB1*”.

To be able to express this kind of effect in the AdDCo algorithm, we extend the language presented in Chapter 4: We introduce an additional construct called the *uniquant*. Furthermore, we redefine the construct *formula*, which can now contain literals as well as uniquants:

$$\mathit{Uniquant} ::= \texttt{(forall } \delta\ l\texttt{)} \qquad \text{where } \delta \in \mathit{Decl}, l \in \mathit{Literal} \tag{5.2}$$

$$\mathit{Formula} ::= \texttt{(and } j_1\ \ldots\ j_n\texttt{)} \qquad \text{where } j_i \in \mathit{Literal} \cup \mathit{Uniquant} \tag{5.3}$$

The functions *vars* and *subst* are defined for uniquants as follows:

$$\mathit{vars}[\![\texttt{(forall } \delta\ l\texttt{)}]\!] = \mathit{vars}\ l \setminus \operatorname{dom} \delta \tag{5.4}$$

$$\mathit{subst}_\beta[\![\texttt{(forall } \delta\ l\texttt{)}]\!] = [\![\texttt{(forall } \delta\ (\mathit{subst}_{\beta'} l)\texttt{)}]\!] \quad \text{where } \beta' = (\operatorname{dom}\delta) \lhd \beta \tag{5.5}$$

Here, $A \lhd R$ denotes a domain anti-restriction, or, more formally,

$$A \lhd R = \{x : X, y : Y \mid xRy \wedge x \notin A \bullet x \longrightarrow y\}$$

The semantics of a forall-term directly follows from its expansion into unquantified terms.

$$\mathit{expand} : \mathit{Dom} \longrightarrow \mathit{Uniquant} \longrightarrow \mathit{Formula} \tag{5.6}$$

$$\mathit{expand}_\tau[\![\texttt{(forall } \delta\ l\texttt{)}]\!] = [\![\texttt{(and } (\mathit{subst}_{\beta_1}\ l)\ (\mathit{subst}_{\beta_2}\ l) \ldots (\mathit{subst}_{\beta_n}\ l)\texttt{)}]\!] \tag{5.7}$$

$$\text{where } \{\beta_1, \ldots, \beta_n\} = \mathit{bindings}\ \delta\ \tau \tag{5.8}$$

Based on this, dq_τ translates a formula to an equivalent quantifier-free formula:

$$dq_\tau[\![\texttt{(and } l_1 \ldots l_n\texttt{)}]\!] = [\![\texttt{(and } (dq_\tau\ l_1)\ \ldots\ (dq_\tau\ l_n)\texttt{)}]\!] \tag{5.9}$$

$$dq_\tau l = l \qquad \text{where } l \in \mathit{Literal} \tag{5.10}$$

$$dq_\tau q = \mathit{expand}_\tau q \qquad \text{where } q \in \mathit{Uniquant} \tag{5.11}$$

so that we arrive at the following modified definition for the effective effect term:

$$\mathit{ceff} : \mathit{Dom} \longrightarrow \mathit{Operator} \longrightarrow \mathit{Formula} \tag{5.12}$$

$$\mathit{ceff}_\tau \omega = \mathit{cff}(dq_\tau(\mathit{eff}\ \omega)) \tag{5.13}$$

A detailed example of instantiating an operator scheme with a universally quantified effect is given in Section 9.2 in the Appendix.

Note that the notation we defined for use with the AdDCo algorithm in the previous chapter and in this chapter is very similar to PDDL. It does, however, not have the full expressivity of PDDL. Our notation allows conjunctions of function-free first-order literals in preconditions, while PDDL supports any first-order logical sentence as a precondition. This does, for example, include disjunctive preconditions, which are not supported by our notation. Furthermore, PDDL effects can be universally quantified and conditional. In contrast, our effects can be universally quantified and conditional only if the condition describes an equality (such as the = term in the example in (5.1)).

5.2 The Persistent Action Problem

Recall that in Section 2.2 we identified the occurrence of *persistent actions* as a characteristic of the smart environments domain. Persistent actions are actions that prevail over a longer timespan, such as the *project* action of a projector. In this section, we show that persistent actions can be modeled elegantly using the AdDCo algorithm. We start with a simple domain description that does not take into account persistent actions. We then show how this description must be altered to model persistent actions correctly if we use planning as a strategy synthesis mechanism. Afterwards, we describe how persistent actions can be modeled if we use the AdDCo algorithm. We finally compare both methods.

A naive way of modelling a smart environment is the following domain *smartenvironment-naive*, described in PDDL [Ghallab *et al.* 1998]:

```
(define (domain smartenvironment-naive)
  (:requirements :strips :equality :typing)
  (:predicates (Pointing ?c - Canvas ?p - Projector)
               (CrossbarIn ?n - Notebook)
               (CrossbarOut ?p - Projector)
               (DocShown ?d - Document ?c - Canvas)
               (isDown ?c - Canvas)
               (Hosts ?d - Document ?n - Notebook)
               (isMax ?d - Document ?n - Notebook)
               (Connected ?n - Notebook ?p - Projector))

(:action CanvasUp
  :parameters (?c - Canvas)
  :precondition (isDown ?c)
  :effect (not (isDown ?c)))

(:action CanvasDown
  :parameters (?c - Canvas)
  :precondition (not (isDown ?c))
  :effect (isDown ?c))

(:action MoveProjector
  :parameters (?c1 - Canvas ?c2 - Canvas)
  :precondition (and (Pointing ?c1 NEC-MT1065 )
                     (not (Pointing ?c2 NEC-MT1065)))
  :effect (and (not (Pointing ?c1 NEC-MT1065))
               (Pointing ?c2 NEC-MT1065)))

(:action SwitchCrossbar
  :parameters (?n - Notebook ?p - Projector)
  :precondition (and (CrossbarIn ?n)(CrossbarOut ?p))
  :effect (and (Connected ?n ?p)
               (forall (?x - Notebook)
                  (when (not (= ?x ?n))
                        (not (Connected ?x ?p))))))

(:action Maximize
  :parameters (?n - Notebook ?d - Document)
  :precondition (Hosts ?d ?n)
  :effect (and (isMax ?d ?n)
               (forall (?x - Document)
                  (when (not (= ?x ?d))
                          (not (isMax ?x ?n))))))

(:action ShowDoc
  :parameters (?n - Notebook ?p - Projector ?d - Document
               ?c - Canvas)
  :precondition (and (Connected ?n ?p)(Pointing ?c ?p)
                     (isMax ?d ?n)(isDown ?c))
  :effect (and (DocShown ?d ?c)
               (forall (?x - Document)
                  (when (not (= ?x ?d))
                        (not (DocShown ?x ?c))))))
```

The domain *smartenvironment-naive* models a smart meeting room containing two notebooks (*NB1*, *NB2*), two documents (*Doc1*, *Doc2*), eight canvasses (*LW1*, *LW2*, *LW3*, *LW4*, *LW5*, *LW6*, *VD1*, *VD2*), three fixed projectors that can each point to one fixed canvas (*EPS3*, *EPS6*, *Panasonic*), and one steerable projector (*NEC-MT1065*) which can point to any of the eight canvasses. This domain can be used for solving the planning problem *presentation*:

```
(define (problem presentation)
  (:domain smartenvironment-naive)
  (:objects NB1 NB2 - Notebook
            Doc1 Doc2 - Document
            LW1 LW2 LW3 LW4 LW5 LW6 VD1 VD2 - Canvas
            EPS3 EPS6 Panasonic NEC-MT1065 - Projector)
  (:init (Hosts NB1 Doc1)
         (Hosts NB2 Doc2)
         (CrossbarIn NB1)
         (CrossbarIn NB2)
         (CrossbarOut EPS3)
         (CrossbarOut EPS6)
         (CrossbarOut Panasonic)
         (CrossbarOut NEC-MT1065)
         (Pointing LW4 NEC-MT1065)
         (Pointing LW3 EPS3)
         (Pointing LW6 EPS6)
         (Pointing VD2 Panasonic))
  (:goal (and (DocShown Doc1 LW3)(DocShown Doc2 LW1))))
```

The goals of the planning problem *presentation* are to show document *Doc1* on canvas *LW3* and document *Doc2* on canvas *LW1*. Using the domain description *smartenvironment-naive*, a planner could generate the following plan:

```
(CanvasDown LW1)
(CanvasDown LW3)
(Maximize NB1 Doc1)
(Maximize NB2 Doc2)
(SwitchCrossbar NB1 NEC-MT1065)
(MoveProjector LW4 LW3)
(ShowDoc NB1 Doc1 NEC-MT1065 LW3)
(SwitchCrossbar NB2 NEC-MT1065)
(MoveProjector LW3 LW1)
(ShowDoc NB2 Doc2 NEC-MT1065 LW1)
```

This plan contains an error. Let us look at the two *ShowDoc* actions: The steerable projector (*NEC-MT1065*) is used to show both *Doc1* on *LW3* and *Doc2* on *LW1*. In the real world, this is not possible, of course. A single projector cannot be used to show two documents on two canvasses simultaneously. Hence, the modelling of the domain is inadequate: Showing a document is a *persistent action*.[1] To be precise, it persists as long as no other action is carried out on the resources it uses. We need to express somehow that if a projector shows a document, it is occupied. As soon as we maximize another document on the same notebook screen, connect the projector to a different computer via the video crossbar, or move the projector to another canvas, the first document is not visible anymore. Hence, the effects of the first *ShowDoc* action are not valid anymore. Thus, there is a dependency between the actions. We call this the *persistent action problem*. It is a key problem in smart environments where persistent actions frequently occur: 5 of the 15 scenarios we analyzed in Chapter 2 contain persistent actions. Furthermore, of

[1]Note that the concept of *durative actions* is very similar, but in the planning community this term is used to denote a feature of PDDL 2.1: It refers to actions which are temporally annotated with an explicit duration, e.g. 5 steps [Fox & Long 2003].

the four scenarios we looked at in Section 4.5.6, three contain persistent actions as well.

The challenge that must be met in smart ad-hoc environments is that the domain description must be assembled at run-time from actions of different origin. Hence, we need to solve the persistent action problem with as little global information as possible. In the following two sections, we describe how to solve the persistent action problem using planning (Section 5.2.1) and using the AdDCo algorithm (Section 5.2.2).

5.2.1 Solving the Persistent Action Problem Using Planning: The Locks Approach

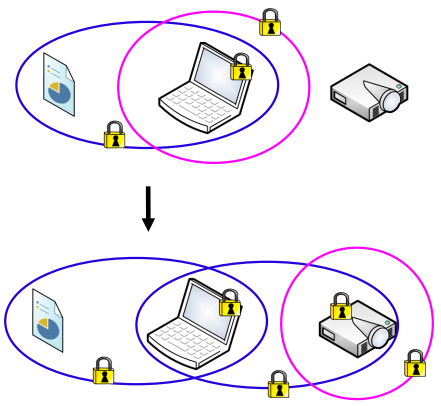
One possibility to solve the persistent action problem in classical planning is to introduce certain literals which we call *locks*. During the planning process, locks prevent chains of actions from being "destroyed" by conflicting actions that use the same resources. Consider the following domain *smartenvironment-locks*:

```
(define (domain smartenvironment-locks)
  (:requirements :strips :typing)
    (:types Notebook Document Projector Canvas - Device)
    (:predicates (isLocked ?d - Device)
                 (isActive ?d1 ?2 - Device)
                 (isConnected ?d1 ?d2 - Device)
                 (Hosts ?d - Document ?n - Notebook)
                 (isDown ?c - Canvas)
                 (CrossbarIn ?n - Notebook)
                 (CrossbarOut ?p - Projector)
                 (Pointing ?c - Canvas ?p - Projector))

  (:action CanvasUp
    :parameters (?c - Canvas)
    :precondition (and (not (isLocked ?c))(isDown ?c))
    :effect (not (isDown ?c)))

  (:action CanvasDown
    :parameters (?c - Canvas)
    :precondition (and (not (isLocked ?c))(not (isDown ?c)))
    :effect (isDown ?c))

  (:action Maximize
    :parameters (?d - Document ?n - Notebook)
    :precondition (and (Hosts ?d ?n)(not (isLocked ?n)))
    :effect (and (isLocked ?n)(isActive ?d ?n)
                 (isConnected ?d ?n)))

  (:action Unlock-Maximize
    :parameters (?d - Document ?n - Notebook)
    :precondition (and (isActive ?d ?n)(isLocked ?n))
    :effect (and (not (isActive ?d ?n))(not (isLocked ?n))
                 (not (isConnected ?d ?n))))

  (:action MoveProjector
    :parameters (?c1 - Canvas ?c2 - Canvas)
    :precondition (and (Pointing ?c1 NEC-MT1065)
                       (not (Pointing ?c2 NEC-MT1065))
                       (not (isLocked ?c1)))
    :effect (and (not (Pointing ?c1 NEC-MT1065))
                 (Pointing ?c2 NEC-MT1065)))

  (:action SwitchCrossbar
    :parameters (?n - Notebook ?p - Projector ?d - Document)
    :precondition (and (not (isLocked ?p))(isActive ?d ?n)
                       (CrossbarIn ?n)(CrossbarOut ?p))
    :effect (and (isLocked ?p)(not (isActive ?d ?n))
                 (isActive ?d ?p)(isConnected ?n ?p)))

  (:action Unlock-SwitchCrossbar
    :parameters (?n - Notebook ?p - Projector ?d - Document)
    :precondition (and (isLocked ?p)(isActive ?d ?p)
                         (isConnected ?n ?p))
    :effect (and (not (isLocked ?p))(isActive ?d ?n)
                 (not (isActive ?d ?p))
                 (not (isConnected ?n ?p))))

  (:action ShowDoc
    :parameters (?p - Projector ?c - Canvas ?d - Document)
    :precondition (and (not (isLocked ?c))(isActive ?d ?p)
                       (isDown ?c)(Pointing ?c ?p))
    :effect (and (isLocked ?c)(not (isActive ?d ?p))
                 (isActive ?d ?c)(isConnected ?p ?c)))

  (:action Unlock-ShowDoc
    :parameters (?p - Projector ?c - Canvas ?d - Document)
    :precondition (and (isLocked ?c)(isActive ?d ?c)
                       (isConnected ?p ?c))
    :effect (and (not (isLocked ?c))(isActive ?d ?p)
                 (not (isActive ?d ?c))
                 (not (isConnected ?p ?c)))))
```

We omit the problem description here as it is the same as in the problem *presentation* described earlier in this section, except for the goal statement, which is now `(:goal (and (isActive Doc1 LW3)(isActive Doc2 LW1)))`.

Three locks are required for every persistent action (see Figure 5.1): The first lock `(isLocked ?d)` locks the resource in question such that no other action can use this resource. In Figure 5.1, this is depicted as a small yellow lock attached to a resource. Thus, the set of locked resources states which resources are currently parts of chains of persistent actions. The second lock `(isConnected ?d1 ?d2)` states which two resources are used consecutively in a chain of actions. In Figure 5.1, this is depicted as a blue ellipse with a little yellow lock attached to it. This lock is important if a new goal is to be fulfilled and this requires that an action sequence previously generated must be unlocked. We will get back to this in Section 5.2.3. The third lock `(isActive ?d1 ?d2)` always denotes the beginning and the current end of a chain (the *tail*). In Figure 5.1, this is depicted as a purple ellipse with a little yellow lock attached to it. During the planning process, this lock is

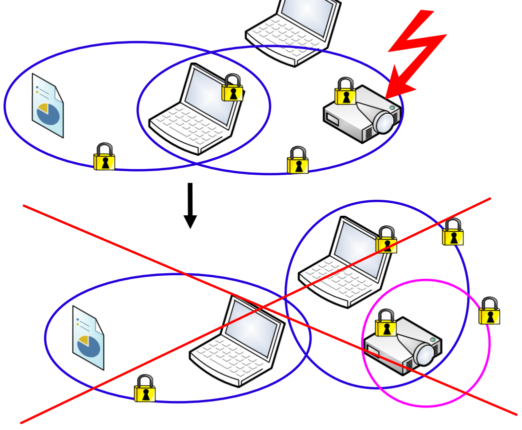
```
(:action SwitchCrossbar
:parameters ()
:precondition (and (not (isLocked NEC-MT1065))(isActive Doc1 NB1)
    (CrossbarIn NB1)(CrossbarOut NEC-MT1065))
:effect (and (isLocked NEC-MT1065)(not (isActive Doc1 NB1))
    (isActive Doc1 NEC-MT1065)(isConnected NB1 NEC-MT1065)))
```



```
(CanvasDown LW3)
(CanvasDown LW1)
(Maximize NB1 Doc1)
(Maximize NB2 Doc2)
(SwitchCrossbar NB1 NEC-MT1065)
(MoveProjector LW4 LW3)
(ShowDoc NB1 Doc1 NEC-MT1065 LW3)
(SwitchCrossbar NB2 NEC-MT1065)
(MoveProjector LW3 LW1)
(ShowDoc NB2 Doc2 NEC-MT1065 LW1)
```

Figure 5.1: One chain locks the resources it occupies...

```
(:action SwitchCrossbar
:parameters ()
:precondition (and (not (isLocked NEC-MT1065))(isActive Doc2 NB2)
    (CrossbarIn NB2)(CrossbarOut NEC-MT1065))
:effect (and (isLocked NEC-MT1065)(not (isActive Doc2 NB2))
    (isActive Doc2 NEC-MT1065)(isConnected NB2 NEC-MT1065)))
```



```
(CanvasDown LW3)
(CanvasDown LW1)
(Maximize NB1 Doc1)
(Maximize NB2 Doc2)
(SwitchCrossbar NB1 NEC-MT1065)
(MoveProjector LW4 LW3)
(ShowDoc NB1 Doc1 NEC-MT1065 LW3)
(SwitchCrossbar NB2 NEC-MT1065)
(MoveProjector LW3 LW1)
(ShowDoc NB2 Doc2 NEC-MT1065 LW1)
```

Figure 5.2: ... so another chain cannot use them.

propagated through the action sequence.

Consider the *SwitchCrossbar* action in Figure 5.1: It has `(isActive Doc1 NB1)` as a precondition. This can be read as "The chain beginning at *Doc1* currently ends at *NB1*". Its effects include `(not (isActive Doc1 NB1))` and `(isActive Doc1 NEC-MT1065)`. I.e., when *SwitchCrossbar* is executed, the tail moves from *NB1* to *NEC-MT1065*. Because every persistent action (apart from *Maximize* which is at the head of the chain) has such a lock as a precondition, the chain beginning with *Doc1* can only be manipulated at its tail. This in conjunction with the `(isLocked ?d)` locks ensures that the chain of actions cannot unintentionally be "destroyed" by a conflicting action. Consider the other *SwitchCrossbar* action in Figure 5.2, which is executed after the first one. This action is part of a second chain beginning with *Doc2*. However, because the `(isLocked NEC-MT1065)` lock is already active, this conflicting action cannot be executed – one of its preconditions is not fulfilled. Hence, the chain beginning with *Doc1* is not broken – the erroneous action sequence in Figure 5.2 cannot be generated. Instead, the locks "force" the algorithm to generate the following correct action sequence:

```
(CanvasDown LW1)
(Maximize Doc2 NB2)
(MoveProjector LW4 LW1)
(CanvasDown LW3)
(Maximize Doc1 NB1)
(SwitchCrossbar NB2 NEC-MT1065 Doc2)
(SwitchCrossbar NB1 EPS3 Doc1)
(ShowDoc NEC-MT1065 LW1 Doc2)
(ShowDoc EPS3 LW3 Doc1)
```

Note that we added a corresponding *unlock* operator for every operator that locks a resource. This enables the planner to unlock the chain starting at its end if new goals are to be fulfilled.

5.2.2 Solving the Persistent Action Problem with the AdDCo Algorithm: The Guarding Approach

The persistent action problem can be solved in a much more elegant way with the AdDCo algorithm. The reason for this is that the AdDCo algorithm is an action selection mechanism. It interleaves planning with execution: Every action selection step is followed by an execution step which changes the world state. This makes it possible to solve the persistent action problem in a fundamentally different way: One can employ the agent (or *CompMod*) of a persistent action A as a “guard” for that action. Guarding means that as long as A is active, A’s agent monitors whether any effect of a subsequent action B that is executed is the opposite of one of A’s preconditions. In this case, it sends a message to all other agents stating that A is not executed anymore and its effects become false. Should one of A’s effects be a precondition of another persistent action C which is currently executed, this process continues: C’s agent will notice that C is not executed anymore, etc.

As an example, reconsider the erroneous action sequence generated by the planner on page 126. This action sequence cannot be generated if we use the guarding approach: Consider the point in the action sequence when the action `(ShowDoc NB1 Doc1 NEC-MT1065 LW3)` is executed (see Figure 5.3). It is a persistent action and requires the literal `(Connected NB1 NEC-MT1065)` to be true. When the action is executed, its agent therefore starts to guard its own preconditions. In Figure 5.3, this is depicted as a small glasses icon above the red dashed line. Now `(SwitchCrossbar NB2 NEC-MT1065)` is executed (see Figure 5.4). This renders the literal `(Connected NB1 NEC-MT1065)` false. The action `(ShowDoc NB1 Doc1 NEC-MT1065 LW3)`’s agent now notices that one of its preconditions is not fulfilled anymore. It notifies the rest of the ensemble that the action is not active anymore and its effects become false. The goal `(DocShown Doc1 LW3)` is now open again and can be fulfilled once more. Hence, the action selection algorithm cannot generate an action sequence that uses the steerable projector *NEC-MT1065* to show *Doc1* on *LW3* and *Doc2* on *LW1* simultaneously. This means the world state in the model of the world does not become inconsistent to the actual world state in the real world. Instead, the correct action sequence depicted in Figure 5.4 is generated. Thus, the guarding approach enables us to model the domain as on page 123.

```
Goals:
(DocShown Doc1 LW3) ✓
(DocShown Doc2 LW1) ✗

(CanvasDown LW3)
(CanvasDown LW1)
(Maximize NB1 Doc1)
(Maximize NB2 Doc2)
(SwitchCrossbar NB1 NEC-MT1065)
(MoveProjector LW1 LW3)
(ShowDoc NB1 Doc1 NEC-MT1065 LW3)
(SwitchCrossbar NB2 NEC-MT1065)
(MoveProjector LW3 LW1)
(ShowDoc NB2 Doc2 NEC-MT1065 LW1)
(SwitchCrossbar NB1 EPS3)
(ShowDoc NB1 Doc1 EPS3 LW3)
```

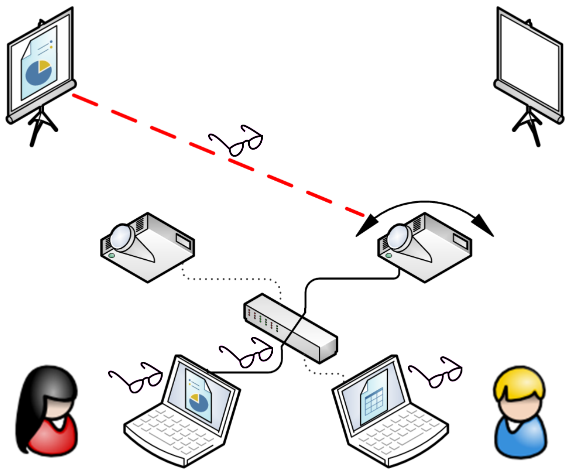


Figure 5.3: The *ShowDoc* action guards its preconditions...

```
Goals:
(DocShown Doc1 LW3) ✗
(DocShown Doc2 LW1) ✗

(CanvasDown LW3)
(CanvasDown LW1)
(Maximize NB1 Doc1)
(Maximize NB2 Doc2)
(SwitchCrossbar NB1 NEC-MT1065)
(MoveProjector LW1 LW3)
(ShowDoc NB1 Doc1 NEC-MT1065 LW3)
(SwitchCrossbar NB2 NEC-MT1065)
(MoveProjector LW3 LW1)
(ShowDoc NB2 Doc2 NEC-MT1065 LW1)
(SwitchCrossbar NB1 EPS3)
(ShowDoc NB1 Doc1 EPS3 LW3)
```

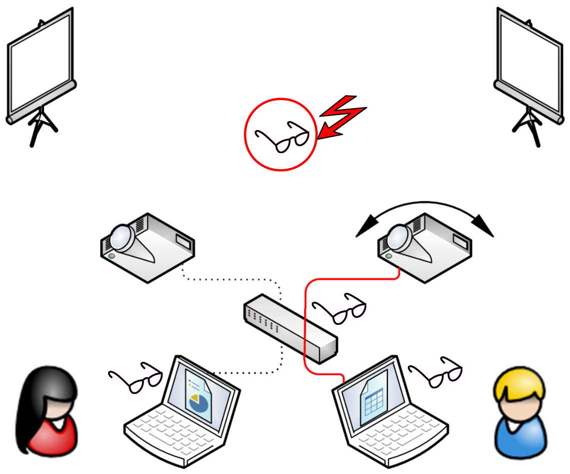


Figure 5.4: ... and notices when one of them is not fulfilled anymore.

Note that without further precautions, the guarding approach can lead to loops where two actions alternately destroy each other's preconditions. In the above example, the actions `(SwitchCrossbar NB1 NEC-MT1065)` and `(SwitchCrossbar NB2 NEC-MT1065)` could be executed alternately again and again. In Section 4.4.10, we have described the mechanism we use in the AdDCo algorithm to prevent such loops: We decrease a CompModInst's probability to become active again each time it is executed.

5.2.3 Comparing the Two Paradigms

In Sections 5.2.1 and 5.2.2 we introduced two paradigms which both solve the persistent action problem. In this section, we elaborate in more detail on the similarities and conceptual differences between the two paradigms. We also point out some of the implications those differences have on modelling and execution of device actions in smart environments.

Both paradigms have in common that actions can be modelled without global knowledge. Each action description can be written using only knowledge about literals that must be fulfilled before the action can be executed and literals that will be true after the action is executed. Of course, it is preferable that the developer of an action description has an idea e.g. which other actions might rely on the effects of the action. This guides the developer's decision which effects to consider for inclusion in the action description. However, the developer need not have knowledge about the complete domain.

One difference between the two paradigms is the cognitive model they resemble: The locks paradigm can be described with the concept of data flow. In the example action sequence shown in Section 5.2.1, the following *isActive* locks become active one after another:

```
(isActive Doc2 NB2)
(isActive Doc1 NB1)
(isActive Doc2 NEC-MT1065)
(isActive Doc1 EPS3)
(isActive Doc2 LW1)
(isActive Doc1 LW3)
```

Doc1 can be seen as data flowing from a source (notebook *NB1*) over an intermediate station (projector *EPS3*) to a sink (canvas *LW3*). Likewise for *Doc2*. In contrast, the agents in the guarding approach resemble the concept of guards that are positioned along a line to the goal, each monitoring whether its assigned persistent action is still being executed. This comes along with a fundamental difference in the approach to solving the persistent action problem: In the locks approach, we prevent conflicting actions from being executed. Thus, the developer has to be careful to add the appropriate locks to the action descriptions of persistent actions. Furthermore, for every operator describing a persistent action, a corresponding unlock operator must be added. This also implies that the overall number of operators is higher. In the guarding approach, on the other hand, we do not prevent conflicting actions from being selected. Instead, for every action A, we monitor whether a conflicting action B terminates A's execution. The developer of an operator for a persistent action must only mark the action as a persistent action. The rest can then be managed by the action selection mechanism.[2]

Another difference manifests itself if an action sequence has been generated and a new goal is to be fulfilled. Consider the point where the action sequence in Section 5.2.1 has been generated and the following new goal arises (the former goals are now not valid anymore): `(:goal (isActive Doc1 LW1))`

This requires the locks approach to execute a number of unlock actions before the new goal can be fulfilled. A planner generates an action sequence such as:

```
(Unlock-ShowDoc NEC-MT1065 LW1 Doc2)
(Unlock-ShowDoc EPS3 LW3 Doc1)
(Unlock-SwitchCrossbar NB2 NEC-MT1065 Doc2)
(Unlock-SwitchCrossbar NB1 EPS3 Doc1)
(SwitchCrossbar NB1 NEC-MT1065 Doc1)
(ShowDoc NEC-MT1065 LW1 Doc1)
```

The existing chain of actions has to be unlocked backwards to the point where necessary actions to fulfill the new goal (*SwitchCrossbar*, *ShowDoc*) can be executed. This is the kind of scenario we need the `(isConnected ?d1 ?d2)` lock for: without it, the planner could not figure out which predecessor an action has

[2] Note that for reasons of simplicity we omitted the handling of persistent actions from the description of our action selection algorithm in Chapter 4. We believe that it is sufficiently straightforward to be reproducible.

and would thus not be able to unlock an existing sequence correctly. The guarding approach does not perform any unlock action in order to fulfill the new goal, it just executes *SwitchCrossbar* and *ShowDoc*. This is both a blessing and a curse. On the one hand, of course, less actions have to be executed. On the other hand, existing action sequences can unintentionally be "destroyed" by conflicting actions because there is no mechanism to protect them.

5.3 Enhancing Design-time Modularization: Hiding CompMods and CompModInsts

Recall that we introduced distributedness as a requirement for strategy synthesis in smart ad-hoc environments in Section 2.2. One reason is that devices in one ensemble are likely to stem from different vendors. If there is a central controlling component, the manufacturer of this component can control how devices of other vendors are being used in the ensemble. Thus, distributing control means distributing the power of vendors. We have called this *design-time modularization*.

Due to its fully distributed nature, the AdDCo algorithm as we introduced it in Section 4.4 already supports design-time modularization to a certain extent. However, it is possible to distribute vendor power even further. With a small change in the AdDCo algorithm, devices can hide any information about the CompMods and CompModInsts they carry from the rest of the ensemble. To realize this, two changes can be performed: First, CompMods do not broadcast their operator schemes as in the original version of the AdDCo algorithm. Instead, they send complete operators over the network. This implies that CompMods do not have to instantiate linked operators as suggested in Section 4.4.6. Second, the operators are sent in a bulk. In other words, each device only broadcasts which preconditions and effects all its operators/CompModInsts have as a whole. The benefit is that CompMods and CompModInsts can be completely hidden by the devices: Other devices in the ensemble never get to know which and how many CompMods and CompModInsts a device has.

As an example, consider the device *Canvas1* in Figure 5.5. Upon entering the ensemble, it introduces itself to the other ensemble members with the following description[3]:

```
(:device Canvas1
    :precondition (and (not (CanvasDown Canvas1))
                       (CanvasDown Canvas1))
    :effect (and (CanvasDown Canvas1)
                 (not (CanvasDown Canvas1)))
```

Links do not connect CompModInsts to other CompModInsts anymore, but CompModInsts to devices. Thus, CompModInsts address messages to another CompModInst not to the CompModInst itself, but to the device carrying it. In fact, a message now has the format <Device name> Precondition|Effect <Precondition>|<Effect> <Message>. Each device has a *secretary* component that distributes all incoming messages to the respective CompModInsts, just like a real secretary distributes incoming calls, faxes and letters to employees (see Figure 5.5). The receivers of a message are those CompModInsts that match the address description, i.e. that have the respective precondition or effect. Consider the following example: CompModInst *Send2Disp* in Figure 5.5 wants to send the amount of energy of 50.23 to its successors. It does not know that *ShowDoc* is its successor. It does not even know how many successors it has. All it knows is that there must be one or more successors on the device *Projector1* because *Projector1*'s device description contains the precondition `(Sent2Disp Document1 Projector1)` which Send2Disp has as an effect. Thus, it sends the following message: *Projector1 Precondition (Sent2Disp Document1 Projector1) Energy 50.23*. This can be understood as "I am sending 50.23 energy units to each CompModInst on device *Projector1* that has the precondition `(Sent2Disp Document1 Projector1)`". *Projector1*'s secretary can now distribute the message to all CompModInsts that match this description. In this case, there is one, namely *ShowDoc*.

Hiding CompMods and CompModInsts clearly has the benefit of enhancing design-time modularization. However, there are some problems with this: Consider the fact that all preconditions and effects of all CompModInsts/operators on a device can now be sent in a single message. However, this device description has to be updated whenever one of the device's CompMods instantiates or deletes

[3]Note that this is not correct PDDL syntax – if this is to be implemented, one has to define an extension to PDDL.

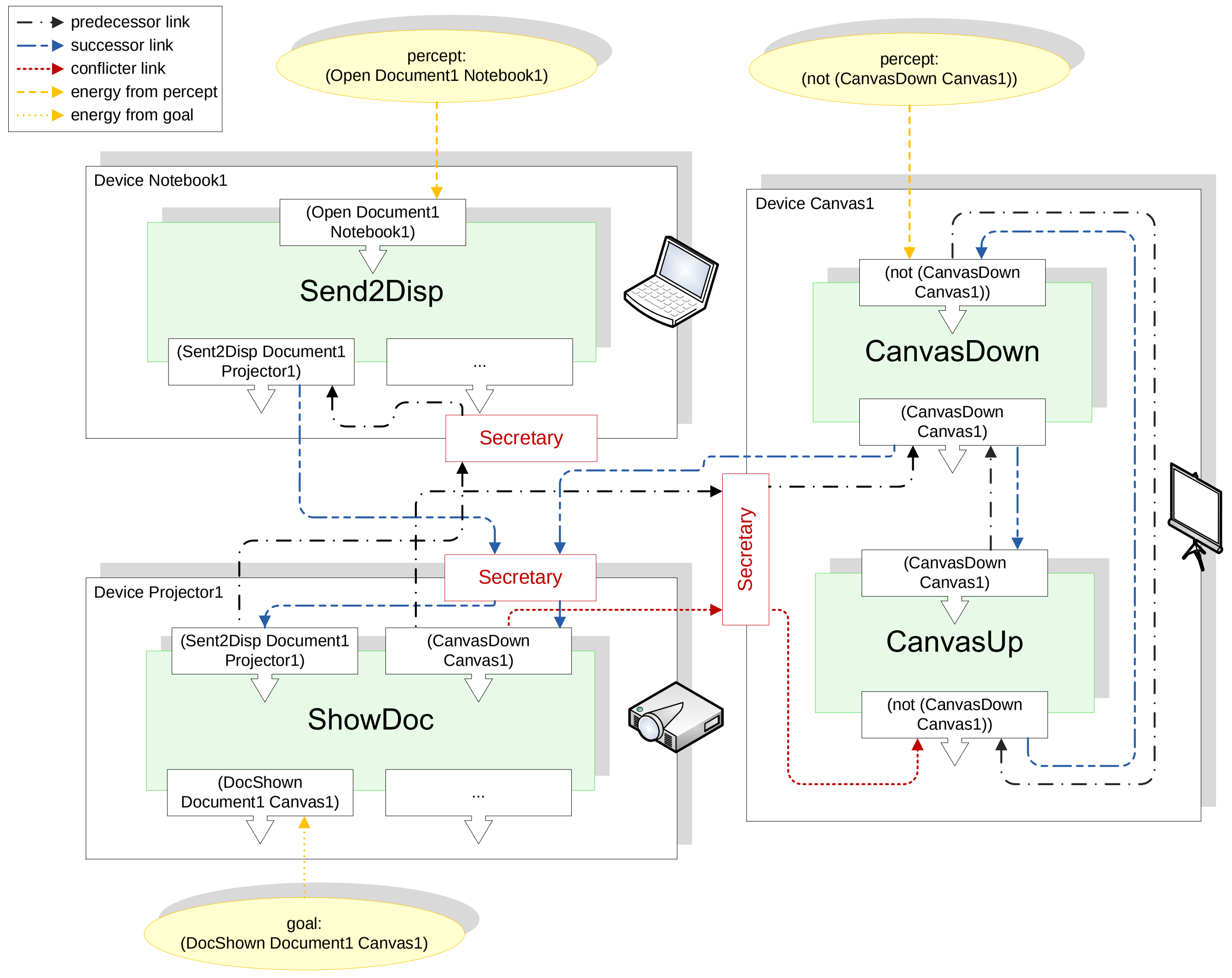


Figure 5.5: Linking CompModInsts via a “secretary”.

CompModInsts, which happens when devices join or leave at run-time. The reason is that instantiating or deleting CompModInsts also creates or deletes preconditions and effects. This creates two problems: First, a device may "guess" some of the internal structures of another device from the adding or removing of preconditions and effects from that device's description. Hence, design-time modularization can be hampered. Second, depending on how many devices join and leave dynamically at run-time, the communication overhead can become bigger than in the original version of the AdDCo algorithm: When hiding CompModInsts, each CompMod has to send a message each time it instantiates or deletes CompModInsts due to the joining or leaving of a device. In the original AdDCo algorithm, only one message per CompMod has to be sent. This message includes the CompMod's operator scheme, which is used by the other CompMods to assemble their linked operators themselves.

5.4 Chapter Summary

In this chapter, we have described three enhancements to the AdDCo algorithm. With those enhancements, the AdDCo algorithm fulfills the following three requirements identified in Section 2.2:

- **Flexibility**: By introducing universally quantified effects as described in Section 5.1, we can express that an effect holds for "All devices of a certain type other than device *x*". Using this construct, authors of action descriptions do not have to know which and how many devices of this type there are in the ensemble. This construct remains the same even when devices join and leave the ensemble. Thus, the AdDCo algorithm now supports dynamic ensemble structures.
- **Support for persistent actions**: Persistent actions are actions that are carried out until they are terminated by another action. In Chapter 2, we have shown that persistent actions occur frequently in smart environments. In Section 5.2, we have shown how persistent actions can be supported in ad-hoc environments: When using planning, the locks approach is feasible. For the AdDCo algorithm, the guarding approach can be used. Both resemble different paradigms, yet both solve the persistent action problem.

- **Distributedness**: The enhancement we introduced in Section 5.3 concerns one part of the requirement *distributedness*, namely *design-time modularization*. The idea is to hide CompMods and CompModInsts inside the devices so that they become "invisible" to the rest of the ensemble. This distributes vendor power as a device manufacturer discloses less information about her/his devices.

Recall that the requirements *spontaneity* and *robustness* were already fulfilled in the last chapter. Now, all requirements from Section 2.2 but one are fulfilled by the AdDCo algorithm: The requirement *rationality* is only partially fulfilled. In the next section, we therefore present a user study that evaluates whether users nevertheless accept an assistance system based on the AdDCo algorithm, and under which circumstances they accept it.

CHAPTER 6

User Study

Contents

In the past chapters, we have introduced the AdDCo algorithm. It is a strategy synthesis mechanism that fulfills all requirements from Section 2.2, apart from the requirement *rationality*. This requirement is not completely fulfilled because the algorithm tends to produce suboptimal solutions. The question is whether users accept such a system. In this chapter, we present a user study that investigates this issue: We strive to find out how different kinds of system behavior influence user acceptance of an exemplary assistance system which is based on the AdDCo algorithm.

Instead of user acceptance, we could evaluate whether the assistance system improves the users' performance in carrying out a certain task. However, we wanted to find out whether people actually intend to use the system, and performance is not a good indicator for this. It is a well-established fact that users may choose not to use a system even if it improves their performance [Davis *et al.* 1989]. We therefore put our emphasis on user acceptance and only look at user performance on the side.

There are two distinct research areas that we draw upon for this study. The first is concerned with people's acceptance of computing systems. Here, scales that measure user acceptance are developed and tested on real systems [Davis *et al.* 1989]. The second is automation science, a field that investigates the effects of automation on human beings, e.g. in domains like flight control or manufacturing plants [Parasuraman 1997, Lee & See 2004, Muir & Moray 1996]. In automation science, systems are "usually large, complex, capital-intensive, and potentially dangerous, and so it is critical they run safely and effectively" [Muir 1994]. Usually, the human operator's task is to monitor the system while it is running and to intervene when necessary in order to maximize safety or productivity. Operators are experts who receive extensive training before starting to work with the machine, and collaboration between the human and the machine takes place in very limited contexts, usually in work settings.

In contrast, assistance systems in Ubiquitous Computing are usually neither capital-intensive nor potentially dangerous, and users are non-experts. Furthermore, the user is most important, while the system is designed to assist her/him. People use such systems not just in one specific setting, but in various contexts. Users are typically not seated in front of a desktop or a display and control panel, but situated in "real life", and typically the user walks into an environment and expects her/his mobile devices to integrate seamlessly with the existing infrastructure. Thus, user acceptance of an assistance system probably depends not only on the system itself, but is likely to be influenced by various contextual factors. It is therefore not clear whether the results from automation research apply to the field of Ubiquitous Computing.

Consequently, we do not only investigate how system behavior influences user acceptance, but take two additional contextual factors into account: whether the user has a low or high task load and whether s/he has experience using the system. The assistance system we use in the study is an instance of a class of systems which is described in Section 6.1.1. As will be shown in Section 6.1.1, our results do not only apply to the particular system we use in the study, but generalize to this class.

The structure of this chapter is as follows: We first review related work and describe the assistance system used in the study. We then present the design of the user study in which 56 participants evaluate the assistance system, describe how we conducted the study and report the results. Using these results, we develop a scheme of user acceptance and performance. We then elaborate on the question which kind of assistance with respect to Wandke's assistance framework [Wandke 2005], which we introduced in Chapter 1, is acceptable for users of Ubiquitous Computing applications. We furthermore discuss some of our findings concerning user interfaces for Ubiquitous Computing assistance systems.

6.1 Related Work

We cover three areas of related work. To motivate that the results of the study generalize to a particular group of applications, we discuss the similarities between the assistance system we use in the study and a specific class of proactive assistance systems. We then present prior research from the field of automation science as a base for the study. Furthermore, we review work on models for evaluating Ubiquitous Computing applications and explain our choice.

6.1.1 Proactive Assistance Systems

In Chapter 3, we have introduced some research projects that have developed assistance systems for smart environments. Of particular interest are those that are flexible enough to generate strategies at run-time. These are typically based on explicit declarative representations of the user's goals. The assistance systems can use these goals to develop a strategy to assist the user. This strategy is not just a mere instantiation of a predefined scheme, but is generated completely at run-time. Here, we briefly recapitulate the projects based on planning we introduced in Section 3.1.2.

In the private household sub-project of the EMBASSI project [Heider & Kirste 2002], the devices carry declarative descriptions (precondition/effect rules) of the actions they can perform and upload them to a central controlling component. When the user utters a goal, the controlling component tries to generate an action sequence for the devices to fulfill this goal. To this end, partial-order planning is used. Once an action sequence has been found, it can be executed autonomously by the device ensemble, without user intervention.

The approach of Amigoni et al. [Amigoni *et al.* 2005] is based on a distributed version of hierarchical task networks called D-HTN (Distributed Hierarchical Task Network). The devices provide descriptions of the actions they are able to perform in the form of HTNs. These are decompositions of higher-level tasks which can be used by a central planning component in order to construct a plan for the whole ensemble. When constructing a plan, the planner can query the available devices for suitable decompositions. In case there is more than one decomposition, the most appropriate one is chosen according to values set by the system designer.

In the O2S approach by Saif et al. [Saif *et al.* 2003], a goal can be viewed as a higher-level function which is to be decomposed into a set of lower-level actions (similar to the HTN approach). There might be several ways to fulfill a goal, and all of those candidates are represented in a goal tree. Choosing the best action sequence to fulfill a goal then corresponds to selecting a path through the goal tree. As in D-HTN, this choice is made according to values specified by the programmer.

In Roadie [Lieberman & Espinosa 2006], a system for assisting the user in a home entertainment scenario, all devices in the home are connected to a central component, supplying descriptions of the actions they are able to perform. As the approaches above, Roadie is based on explicit user goals. Using these goals, Roadie can generate a plan using the Graphplan planner [Blum & Furst 1995]. Some of these actions it can perform on its own, while it instructs the user on how to accomplish the rest.

All of these systems share the following features with the assistance system we use in the study, which is based on the AdDCo algorithm introduced in Chapter 4:

- They are based on explicit declarative user goals. Hence, they abandon the

functional paradigm in favor of the *goal-based* paradigm, which is much more user-friendly. We have introduced goal-based interaction in Section 3.1.2.
- They generate strategies at run-time, which makes them suitable for environments with a dynamic device infrastructure.
- They are inherently imperfect. This is due to the run-time strategy synthesis techniques they employ: Planning can yield suboptimal solutions for all but the most trivial problems, and the same applies for the AdDCo algorithm.

Due to these commonalities, we argue that the results of the user study generalize to the class of systems discussed above.

6.1.2 Research on the Effects of Automation

The papers we discuss in this section are from the area of automation science. Here, *trust* is a commonly used concept to characterize user acceptance. This is due to the fact that system failures can be expensive or even dangerous to the operator. To give the reader an understanding of the concept of trust in this particular context, we therefore present two definitions before discussing the papers. Lee and See give a rather general definition [Lee & See 2004]:

> "Trust can be defined as the attitude that an agent will help achieve an individual's goals in a situation characterized by uncertainty and vulnerability."

Muir and Moray present a more sophisticated model of trust that is tailored to automation [Muir & Moray 1996]:

> "[Trust] in automation is a composite expectation of (1) the operator's general expectation of the *persistence* of the natural physical order, the natural biological order and the moral social order, (2) a specific expectation of the *technical competence* of the automation and (3) a specific expectation of the *fiduciary responsibility* of the automation."

Parasuraman and Riley review automation science papers to find out what influences people's decision to use or not use automation [Parasuraman 1997]. They find that the user's attitude towards a machine depends strongly on the reliability of the machine in some cases, in other cases no such dependence could be found. This issue has not been fully explored yet. Whether a higher workload leads to greater automation use is also unclear. It seems to depend on the person and on the task and must be studied separately for each domain. If the user does not perceive the advantage automation offers as being sufficient to overcome the overhead associated with setting it up, s/he may not use the automation. Occasional automation failures do not necessarily lead to less automation usage in the future. This depends on:

- whether the automation is usually reliable,
- whether system behavior and system state are transparent to the user,
- the overhead involved in turning the automation on or off, or
- the complexity of the task.

When speaking about reliability, one has to bear in mind that people's expectations of reliability depend strongly on the domain. As an example, consider different modes of transport: Obviously, people expect a much higher reliability when they fly by plane than when they go by car. To the author's knowledge, it has not yet been investigated which level of reliability people expect from assistance systems in ubiquitous computing. However, consider the fact that these systems are usually not safety-critical. Therefore, we can assume that people's reliability expectations are lower here than for planes or trains. A reasonable guess is that people expect such systems to be about as reliable as traditional computing applications.

Itoh et al. investigate how trust and use of automation depend on occurrence patterns of malfunctions [Itoh *et al.* 1999]. They find that if a series of malfunctions occurs, trust is affected more than if malfunctions occur occasionally. If operators are experienced, trust recovers more quickly after three successive malfunctions than for unexperienced operators. Experienced operators also tend to rely on automation if they perceive a small error, while less experienced operators tend to switch to manual control in such situations.

According to Lee and See, people's reliance on automation depends on trust if the system is too complex to be fully understood or if the situation demands for some creativity [Lee & See 2004]. If a person understands the algorithms behind a system or believes that the system can fulfill her/his goals, s/he tends to trust the system. New users base their trust on the available information about the system's purpose, while later on users develop a feeling for the system's reliability and predictability. If a user believes that a system functions correctly and is disappointed, s/he may choose not to use the system in the future. Self-confident users who do not trust a system very much tend to manual control. The opposite is also true: People with little self-confidence tend to rely on automation more often. A small error with non-predictable results influences trust more than a constant larger error. People tend to rely on faulty automation if they know in advance which faults can occur. On the other hand, if information about the functionality of a system is not available or displayed improperly, trust can not develop appropriately. Lee and See also point out that system designers should not try to evoke maximum trust in a system, but a level of trust that is appropriate for the system's capabilities.

Muir and Moray investigate how trust relates to human intervention in a process control simulation [Muir & Moray 1996]. They find that trust in a machine depends strongly on how competent people perceive this machine to be, and that trust is strongly correlated with automation usage. They furthermore find that control and display errors of a simulated pump affect trust in an additive fashion. A small variable error has a similar effect as a relatively large constant error. If a system has an automatic and a manual mode, people use the automatic mode more often the more they trust the machine. The less an operator trusts a machine, the more will s/he monitor it. However, Moray and Muir also find that trust does not go down to 0 if small errors are encountered.

As said before, in contrast to the systems under research in automation science, Ubiquitous Computing systems are usually not large, expensive, and safety-critical, and users are no trained experts. Therefore, we believe that trust should not be the primary concept to capture user acceptance in this field. Questions like "Does the user think that the assistance system is useful?" or "Does the user find the assistance system easy to use?" are more likely to be related to user acceptance of assistance systems in Ubiquitous Computing. We therefore discuss different models that capture such concepts in the next section.

6.1.3 Models for Evaluating Ubiquitous Computing Applications

To develop a test for user acceptance is a research project of its own. We therefore decided to look at existing models, identify the most suitable model for our purpose, and use the scales from that model.

Scholtz and Consolvo present a framework for evaluating Ubiquitous Computing applications according to nine dimensions: attention, adoption, trust, conceptual models, interaction, invisibility, impact and side effects, appeal, and application robustness [Scholtz & Consolvo 2004]. This framework provides a good understanding for what the important issues in evaluating Ubiquitous Computing applications are. However, it is rather intended as a concept to guide Ubiquitous Computing researchers in developing the design of user studies than as an actual test for user acceptance. It does not include any concrete items that could be used in a questionnaire.

The scales from the Technology Acceptance Model (TAM) [Davis 1989] proposed by Davis have been applied in many domains, e.g. knowledge management systems [Money & Turner 2004] and code inspection systems [Laitenberger & Dreyer 1998]. TAM has been referenced in 450 publications. It consists of two constructs: perceived usefulness (PU) and perceived ease of use (PE). Both are measured using six-item scales. TAM is a predictor of people's actual usage behavior, i.e. an application with a high PU and PE value is likely to be used.

Several researchers have adapted TAM for Ubiquitous Computing. Connelly proposes a version of TAM for pervasive computing called PTAM [Connelly 2007]. It includes more constructs than TAM: In addition to the constructs PU and PE from Davis' TAM, PTAM comprises Social Influence, Trust, and Integration. According to Connelly, the three additional constructs are important in Ubiquitous Computing. Connelly does not propose concrete items to measure the five constructs, but states that six items are necessary for each construct to obtain sufficient reliability of the scales. Thus, 30 items are required to assess the complete PTAM. For logistic reasons, we could not use PTAM: In our study, each participant was presented three scenarios and had to fill out a questionnaire after each scenario. Had we included 30 items into the questionnaire, each participant would have rated 90 items altogether. This would have overburdened the participants. For the same reason we did not use UTAUT, the Unified Theory of Acceptance and Use of Technology [Venkatesh *et al.* 2003]. It is newer and more comprehensive than TAM, but, like PTAM, has too many items.

Spiekermann proposes UC-AM, the Ubiquitous Computing Acceptance Model

[Spiekermann 2008]. It comprises six constructs: Perceived Usefulness, Perceived Ease of Use, Intention to Use, Intention to Buy, Affective Attitude, and Perceived Control. The validation of the model consists of scenarios of future applications that were rated by participants in on-line and paper surveys. Thus, it is not sure whether UC-AM is also applicable when evaluating existing Ubiquitous Computing applications.

Besides its compactness, the most convincing argument in favor of TAM is the following: The studies cited in this section have found PU and PE to be the main determinants of user acceptance. Other factors have minor influence. This indicates that TAM is a good choice. We therefore decided to use the PU and PE scales from the original TAM in a slightly adapted version: We omit references to jobs as Ubiquitous Computing applications are not only present in work contexts, but in many areas of daily living. Table 6.1 lists the resulting PU and PE scales. To ensure that they are sufficiently reliable, we calculated Cronbach's α using the answers from the questionnaires the participants had to fill out during the user study.[1] Cronbach's α is 0.88 for the modified PU scale and 0.76 for the modified PE scale. Both values are above 0.7 which indicates that the scales are reliable.

Perceived Usefulness (PU)
Using the assistance system enables me to accomplish tasks more quickly.
Using the assistance system improves my performance.
Using the assistance system increases my productivity.
Using the assistance system enhances my effectiveness.
Using the assistance system makes it easier to complete my tasks.
I find the assistance system useful.
Perceived Ease of Use (PE)
Learning to operate the assistance system is easy for me.
I find it easy to get the assistance system to do what I want it to do.
My interaction with the assistance system is clear and understandable.
My interaction with the assistance system is flexible.
It is easy for me to become skillful at using the assistance system.
I find the assistance system easy to use.

Table 6.1: The modified PU and PE scales from TAM. All 12 items are to be ranked on a five-point Likert scale[2], where 4 corresponds to "I totally agree" and 0 to "I do not agree at all".

[1]Cronbach's α is a measure that indicates how reliable a scale consisting of multiple items is, i.e. to which degree those items measure a common construct. More information can be found in [Bortz & Döring 2006].

[2]A Likert scale is a scale commonly used in questionnaires. It often consists of five or seven points

6.2 The Assistance System Used for the Study

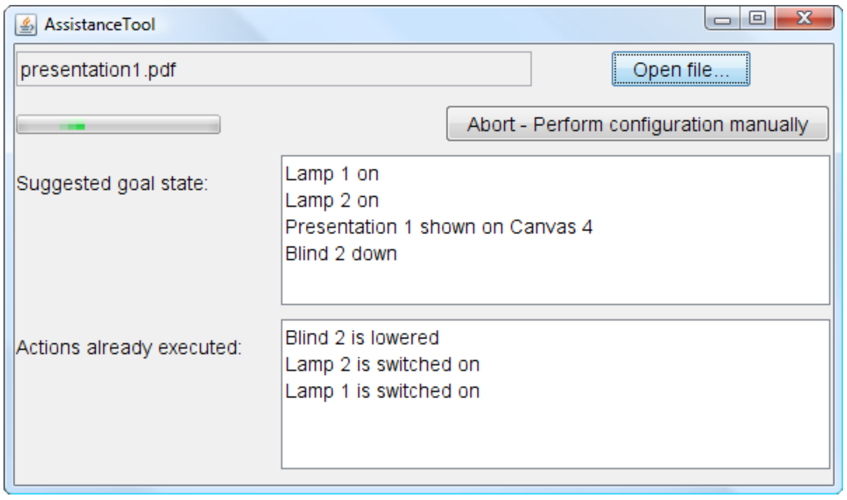
The assistance system we used for the study is based on the two-stage approach to user assistance we introduced in Section 1.2: It conceptually consists of the intention analysis and the strategy synthesis. Finding out which goals the user has and fulfilling those goals can both cause errors. In the user study, however, we wanted to concentrate on the errors that can arise when trying to fulfill the goals. Therefore, the assistance system contains only the strategy synthesis; we omit the intention analysis and specify the user's goals in advance instead.

The strategy synthesis mechanism employs the AdDCo algorithm for action selection. It takes user goals as an input and generates an action sequence that fulfills the goals. As soon as an action has been chosen, it is executed by the devices. As the algorithm operates without global knowledge and employs no predefined schemes, the action sequences generated can be suboptimal. Apart from the fact that it is fully distributed, this system exhibits similar properties to those of the proactive assistance systems described in Section 6.1.1.

As discussed in Section 4.4.12, the AdDCo algorithm breaks ties randomly. Thus, different runs of the algorithm may yield different action sequences for the same input. For the study, however, we needed to make sure that all participants would be presented the same action sequences. Therefore, we used a mockup system that behaved exactly like the real system. It replayed a typical action sequence generated by the AdDCo algorithm during a run prior to the study. Furthermore, the mockup system took exactly as long as the real system to find this sequence, so that participants did not notice it was a mockup.

According to Parasuraman and Riley, people tend to accept suboptimal automation if they receive adequate feedback about the system state [Parasuraman 1997]. We therefore created a graphical user interface that allows participants to control and monitor the assistance system (see Figure 6.1). It displays the goals the system is currently trying to fulfill and descriptions of the actions that have been executed so far. According to Muir, people accept a system better if they have the possibility to override decisions of the system [Muir 1994]. We thus included a manual mode of operation which participants could invoke if they were not satisfied with the behavior of the automatic assistance (see Figure 6.2). The manual mode is a

(or levels) and assesses to which degree the participant agrees to a statement. More information can be found in [Bortz & Döring 2006].



Suggested goal state:

Lamp 1 on
Lamp 2 on
Presentation 1 shown on Canvas 4
Blind 2 down

Actions already executed:

Blind 2 is lowered
Lamp 2 is switched on
Lamp 1 is switched on

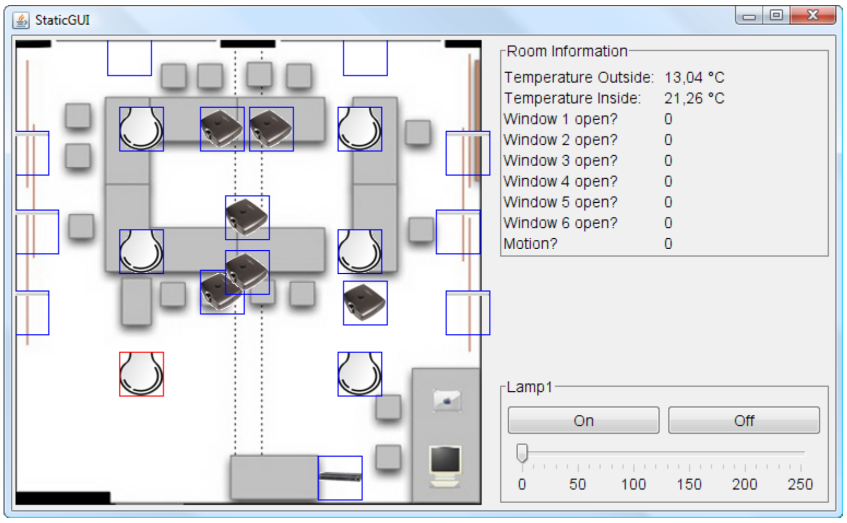
Figure 6.1: The user interface of the assistance system.

user interface where all devices in the room are represented as icons and can be controlled via point-and-click interaction. It is an easy to use version of the kind of room control panel installed in many of today's lecture rooms. Both user interfaces ran on a notebook the participants were given for the study.

We have just argued that for a system to be accepted, it is crucial that users can monitor its execution and override its actions. This implies that users are aware of the assistance. Note that this does not conflict with the assistance being *unobtrusive*. Remember that in Section 2.3, we discussed that unobtrusiveness does not imply that the user is unaware *that* s/he is being assisted, but that s/he should not have to know *how* s/he is being assisted.

Recall Wandke's assistance framework [Wandke 2005] which we introduced in Section 1.3. The assistance system we used in this study can be classified according to the three dimensions as follows:

- **The stages of human-machine interactions that can be assisted**: This system supports the stage *decision making/action selection*, where it provides *informative execution assistance*. All actions are executed by the system automatically.
- **Adjustment**: This system provides *fixed assistance*. The same assistance system is used by all users in all situations.
- **Initiative**: This system provides *active assistance*. Assistance is initiated by the system.


StaticGUI

Room Information

```
Temperature Outside:  13,04 °C
Temperature Inside:   21,26 °C
Window 1 open?        0
Window 2 open?        0
Window 3 open?        0
Window 4 open?        0
Window 5 open?        0
Window 6 open?        0
Motion?               0
```

Lamp1

Figure 6.2: The user interface for manual configuration.

Our hypothesis in Section 2.3 was that users accept a system with these properties. Thus, by investigating *whether* our assistance system is acceptable to the users and under which *circumstances* it is acceptable, we will be able to draw some general conclusions whether our hypothesis holds.

As mentioned before, one of those *circumstances* we have in mind is system behavior. It all boils down to the question "Will users accept an assistance system even if it exhibits suboptimal behavior?", or, in other words, "Does an assistance system have to act fully *rational* to be accepted?" This is of particular interest. To see why, let us look at two quotes from the literature:

> "If the assistance of the system is violating the expectations of the user and forcing him to correct the system manually, he will dislike the system." [Heider 2010]

> "[A] context-aware artifact may fail annoyingly as soon as a context-aware system's (wrong) choices become significant." [Lueg 2002]

Two things are apparent from the quotes: First, Ubiquitous Computing research often treats system behavior simply as a binary variable with the outcomes *success* or *failure*. We think this is not an adequate characterization and will thus offer a refinement of the notion of system behavior in the next section: We introduce the *four levels of imperfection*. Second, Ubiquitous Computing research is largely guided

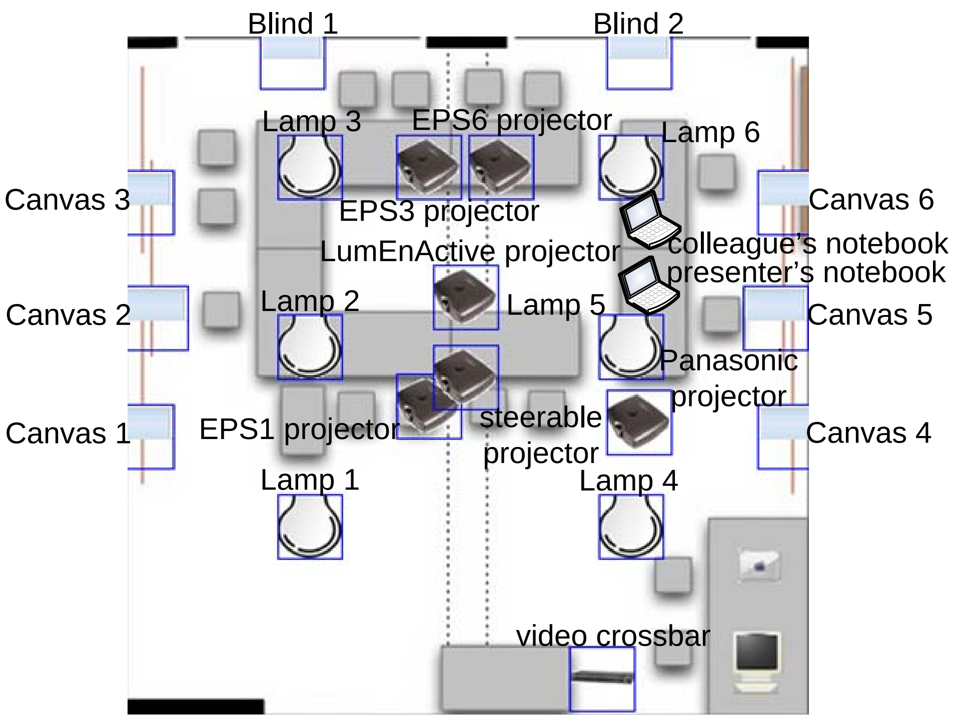


Figure 6.3: The technical infrastructure used in the experiments: All devices depicted here could be operated by the automatic assistance or manually by the users.

by the assumption that assistance systems should always behave sensibly if users are to accept them, and that users will be annoyed if an assistance system exhibits suboptimal behavior. In contrast, our hypothesis is that users accept suboptimal system behavior, provided that they can easily correct this behavior if required.

6.3 Conducting the User Study

6.3.1 The Design of the Study

The experiments were conducted in a smart environment containing the following devices: six lamps, six canvasses which can be lowered and raised, two blinds in front of the windows which also serve as canvasses, six projectors (one of which is steerable and can project onto all canvasses), and a video crossbar. For the experiments, we added two notebooks as mobile devices. The room layout is depicted in Figure 6.3.

In the study, we measured the influence of three factors on user acceptance of the assistance system. As discussed in the last section, one possible determinant of user acceptance is system behavior, which has also been investigated e.g. by Muir and Moray [Muir & Moray 1996] in automation science. We thus included system behavior as a factor. However, for Ubiquitous Computing systems one also has to take the context into account: The situation the user is in may influence user acceptance. To measure the influence of the user's current situation, we included task load as a factor, while situation in a longer-term sense is captured by the factor experience. We now discuss the three factors in detail.

System Behavior Any system that assists a user can behave more or less imperfect or, in other words, can act more or less rationally. To be able to classify such behavior, we introduce the following *four levels of imperfection*:

- **Level 1** – Directly achieving the user goals: The system provides perfect assistance.
- **Level 2** – Eventually achieving the user goals: The system behaves in an unexpected way, but nevertheless fulfills the user's goals. This may irritate a user.
- **Level 3** – Doing nothing: The system does not perform any action. The user must configure the devices manually and loses time.
- **Level 4** – Doing the wrong thing: The system behaves in an unexpected way and does not manage to fulfill the user's goals. Thus, the system hinders the user as s/he must undo any unwanted actions and configure the devices her-/himself.

We developed the categorization into these four levels of imperfection while assessing different mechanisms for user assistance in our lab over several years. We found that these four levels of behavior are reoccuring patterns which are not specific for a single assistance system. Therefore, we believe that they are suited for characterizing different kinds of desired and undesired system behavior.

While experiencing Level 2 assistance, users can usually not tell whether it is Level 2 or Level 4 behavior until the goals have been fulfilled. The assistance system we use for the study exhibits behaviors belonging to Level 1, 2, and potentially even 4, depending on the complexity of the scenario. In this study, we wanted to find out how participants accept Level 1 and Level 2 behavior in an assistance system. We thus created one scenario with Level 1 behavior and two scenarios with

Level 2 behavior. The reason why we chose these scenarios is that they reflect typical usage situations we experienced in our lab. From all potential usage situations, we selected those for the study that represent different yet typical kinds of system behavior. The differences in system behavior stem from different levels of difficulties of the tasks (i.e. the goals to be fulfilled). Here it must be remarked that "difficult" from the perspective of a human does not necessarily mean "difficult" for the system and vice versa.

In all scenarios, the participants assume the role of a presenter who walks into a meeting room equipped with a high number of devices and has to configure this room for a talk using her/his notebook. All three scenarios start in the automatic mode, but participants can switch to manual mode at any time.

Scenario 1 (Level 1 behavior): The presenter's goals are to switch on Lamp 1 and Lamp 2, lower Blind 2, and show her/his presentation, Presentation 1 (which is in PDF format), on Canvas 4. In the automatic mode, the assistance system finds the optimal (shortest) action sequence consisting of eight actions to fulfill the goals. The participant need only wait for the system to finish configuring the devices in the room, which takes 54 seconds. Thus, these goals are easy to fulfill for the system; it exhibits Level 1 behavior. If the subject switches to manual control, s/he has to turn on Lamp 1 and Lamp 2, lower Blind 2 and Canvas 4, turn on the steerable projector, steer it to Canvas 4, and connect the video signal from the notebook to the steerable projector via the video crossbar. All of this is to be done using the manual interface (see Figure 6.2). Then s/he has to open the presentation in a PDF viewer on her/his notebook and maximize it.

Scenario 2 (Level 2 behavior): This scenario is similar to Scenario 1, but the presentation is in PPT format. The participant's notebook has a PDF viewer, but no PPT viewer installed. However, a colleague has a PPT to PDF conversion service running on her notebook and offers to use it. In the automatic mode, the assistance system manages to send the PPT file to the colleague's notebook automatically, converts it using the conversion service and sends back the generated PDF file to the participant's notebook, which can then display it. In this scenario, the goals are more difficult to fulfill for the system, hence the Level 2 behavior: The assistance system takes 74 seconds to find an action sequence consisting of twelve actions, where the optimum is eleven. Thus, the automatic assistance performs one unnecessary action: It opens the converted PDF document on the colleague's notebook.

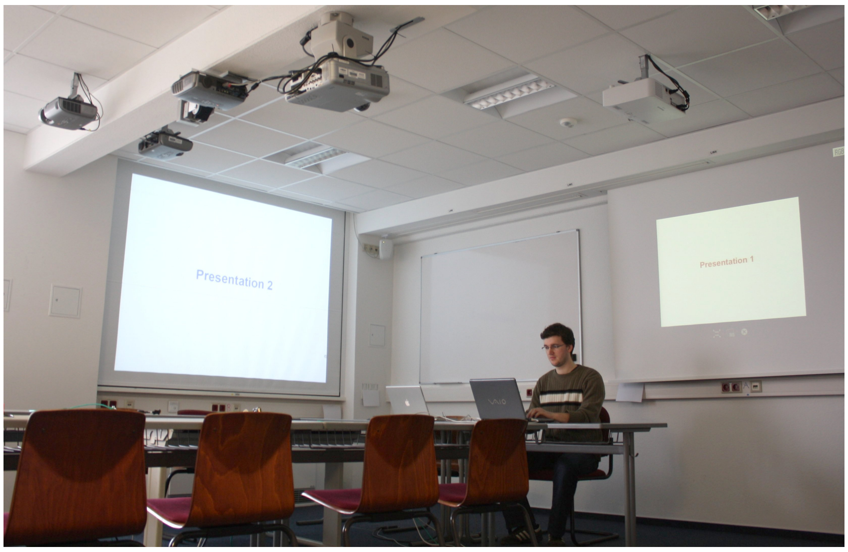


Figure 6.4: A participant in Scenario 3.

If the participant switches to manual control, s/he must perform the same actions as in the first scenario. S/he must also transfer the PPT file to the colleague's notebook using a USB stick, open the PPT file in the PPT viewer, export it to PDF and copy the PDF file to her/his notebook, once again using the USB stick. Then s/he can display the PDF presentation on her/his notebook using the PDF viewer.

Scenario 3 (Level 2 behavior): The presenter's initial goals are the same as in Scenario 1. When they have been fulfilled, someone from the audience (played by the experimenter) asks a question. The answer can be given by showing a diagram from a another presentation, Presentation 2, which is on the presenter's notebook. Thus, Presentation 2 should be shown on Blind 2, while Presentation 1 should remain visible on Canvas 4. A colleague offers to use his notebook for displaying one of the two presentations. In the automatic mode, the assistance system manages to show both presentations, but takes 119 seconds and performs 30 actions until all goals are fulfilled, where twelve is the optimum. From the fact that the system exhibits Level 2 behavior with 18 unnecessary actions, one can see that the goals in this scenario are rather hard to fulfill for the system. The system takes 30 actions because the layout of the devices in the room is such that the fixed Panasonic projector is pointing to Blind 2, but no fixed projector points to Canvas 4, so the steerable projector *must* be used to display something on Canvas 4. The automatic assistance first displays Presentation 2 on Blind 2 via the steerable projector. Then it "realizes" that no projector is left to display Presentation 1 on Canvas 4. It then turns the steerable projector to Canvas 4 and displays Presentation 1, but now of course Presentation 2 is not visible anymore, so the steerable projector is turned back onto Blind 2. After a while the assistance system "realizes" that it must use the fixed Panasonic projector to display Presentation 2 on Blind 2 and the steerable projector to display Presentation 1 on Canvas 4 (see Figure 6.4).[3] If the subject switches to manual mode, s/he must perform the same actions as in the first scenario. S/he must then copy one presentation to the colleague's notebook via the USB stick and display it in the PDF viewer. Finally, s/he must connect the video signals from the two notebooks to two projectors via the video crossbar using the manual interface.

Task Load Ubiquitous Computing systems are designed for a variety of situations, e.g. to help people configure environments within a limited amount of time, possibly in front of other people. For example, when configuring a meeting room s/he has never used before for a talk, the user may not know the devices in the room. S/he may have secondary tasks such as configuring the headset, and may be nervous due to the upcoming talk. In other words, the user may use such a system when experiencing a high level of task load. In many situations, this can lead to stress. However, stress is a much broader concept than task load. To determine a person's stress level, it is necessary to measure physiological parameters such as blood pressure or skin resistance. We could not do this in the study. In the following, we will therefore speak about task load rather than stress.

To assess how task load influences user acceptance, we gave half of the participants a secondary task. We call these these participants the *dual task group* in the following, while we call the other half the *single task group*. Next to configuring the room, the dual task group had to solve simple arithmetic tasks, e.g. $668 - 356$ or $124 : 3$. This is a widely used method to increase task load [Van Gemmert & Van Galen 1997, Castro *et al.* 2009]. To give participants a motivation to solve the arithmetic tasks on the one hand and to finish configuration of the room quickly on the other hand, the amount of compensation the dual task

[3]During the experiments it turned out that people have similar problems here as the automatic assistance. Four of the participants that switched to manual mode in Scenario 3 tried the steerable projector with Blind 2 before realizing they must use the fixed Panasonic projector for Blind 2 and the steerable projector for Canvas 4.

group received for taking part in the study depended on how they acquitted themselves. Initially, their balance was 9 Euros. For every arithmetic task they solved incorrectly or not at all, 20 Cents were deducted. For every minute the experiments took, 50 Cents were deducted. These values were chosen so that most people from the dual task group would receive between 5 and 6 Euros. Each participant from the single task group received a fixed amount of 5 Euros.

Experience Acceptance of an assistance system is not static, but evolves with the experience the user gains when using the system. We propose to describe this process with three phases:

- **Phase 1**: First impression of the system.
- **Phase 2**: Some experience using the system.
- **Phase 3**: Long-term experience using the system.

In this study, we were interested to find out how Phase 1 and Phase 2 influence user acceptance of an ad-hoc assistance system because these are the typical usage situations for this kind of application. Phase 1 corresponds to a situation where the user walks into a smart environment and is confronted with a completely new situation, e.g. s/he has never been to this meeting room before and has to configure it using the assistance system. Phase 2 corresponds to a situation where the user has some time to get acquainted with the assistance system, e.g. s/he is the first speaker in a conference session, the audience has not yet arrived and the user has some time to try out the infrastructure in the room. It would also be interesting to investigate how Phase 3 influences user acceptance, but we refrain from it for two reasons: First, a longitudinal study would have to be carried out, which would require considerably more time and resources, especially with the number of participants we were aiming at. Second and most important, systems like ours are walk-up-and-use systems. In practice, the device configuration in the environment and thus the assistance system itself will most likely have changed before Phase 3 is reached.

To assess how experience influences user acceptance, half of the participants were allowed to familiarize themselves with the system in a training phase (Phase 2), the other half were not (Phase 1). To keep the training phase short, the experimenter would first demonstrate the automatic and the manual mode before allowing the participant to try out the system her-/himself. The participant could operate a few devices her-/himself via the manual interface until s/he felt confident using the system. The training phase usually took five to seven minutes.

Summary of the Study Design In summary, we have three factors: system behavior (three levels), task load (two levels), and experience (two levels) in a fully crossed 3x2x2 design with twelve cells (see Figure 6.5). To have a good chance of getting significant results in the statistical tests we were planning, we needed at least 14 samples for each cell, which is 168 samples altogether.[4]. We treated task load and experience as between-subjects factors: Each participant was put either in the single task group or in the dual task group, and would either receive training or no training. In contrast, we treated system behavior as a within-subjects factor: Each participant was presented all three scenarios. Thus, we needed 2x2x14 = 56 participants.

		scenario 1	scenario 2	scenario 3
without training	single task	14 participants		
		14 experiments	14 experiments	14 experiments
	dual task	14 participants		
		14 experiments	14 experiments	14 experiments
with training	single task	14 participants		
		14 experiments	14 experiments	14 experiments
	dual task	14 participants		
		14 experiments	14 experiments	14 experiments

Figure 6.5: The study design in a matrix.

To check whether manipulation of the factors system behavior, task load, and experience had an influence on the results, we included one manipulation check for each factor. After each trial, people were asked to rate how reliable they found the assistance system on a five-point Likert scale. This was to check if the quality of the action sequence found by the system had any effect on how reliable people perceived the assistance system. After the last trial, we asked people to rate the level of task load they experienced for each trial and how competent they felt in using the assistance system, again on a five-point Likert scale. The former was to check whether the arithmetic tasks had a significant effect on people's task load level, while the latter was to find out if people in the training group felt that they had more experience than those that were not trained.

[4]For details on calculating the sample size, see [Bortz & Döring 2006], p. 627ff.

6.3.2 Experimental Procedure

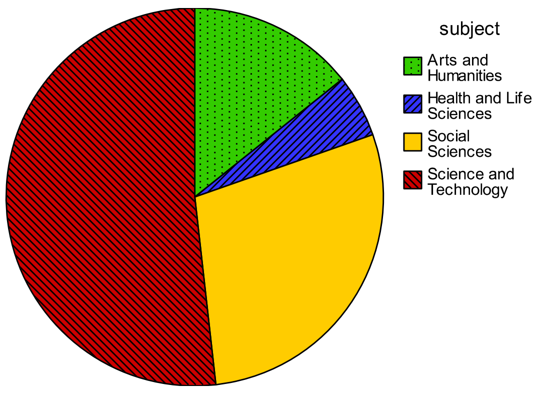
The user study was conducted in eight days and consisted of 56 sessions – one for each participant. Each session took about 30 minutes and consisted of three trials corresponding to the three scenarios. Thus, we conducted 168 individual trials. As each participant was presented all three scenarios, we conducted 56 trials for each scenario. As half of the participants were trained, we conducted 84 trials with training and 84 without. As half of the participants had a secondary task, we conducted 84 dual task and 84 single task trials (see Figure 6.5).

In the beginning of the session, each subject completed a questionnaire that assessed the control variables: demographic data and the TA-EG scale for measuring technophilia [Karrer *et al.* 2009]. This scale consists of 19 items to be rated on a five-point Likert scale and is depicted in Table 6.2. The technophilia value of a person was calculated by summing up the participants' ratings for each item and normalizing this sum to a range of 0 (technophobic) to 4 (technophilic).

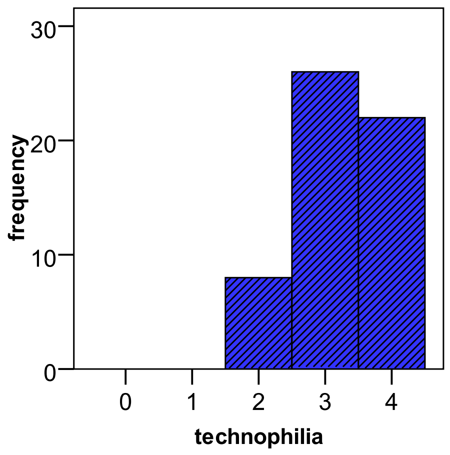
I stay informed about electronic devices even if I do not intend to buy any.
I love possessing new electronic devices.
I am thrilled if a new electronic device is released.
I like browsing shops for electronic devices.
I enjoy trying out electronic devices.
I know most of the features of the electronic devices I own.
I (would) have problems understanding magazines about electronics/computers.
I find it easy to learn to operate an electronic device.
I know a lot about electronic devices.
Electronic devices help to gather information.
Electronic devices allow for a high standard of living.
Electronic devices enhance safety.
Electronic devices make me independent.
Electronic devices facilitate my daily life.
Electronic devices reduce personal contact between people.
Electronic devices cause stress.
Electronic devices make people sick.
Electronic devices make many things more complicated.
Electronic devices lead to mental impoverishment.

Table 6.2: The TA-EG scale for assessing technophilia. All 19 items are to be ranked on a five-point Likert scale. For the first 14 items 4 corresponds to "I totally agree" and 0 to "I do not agree at all", while for the last five items 4 corresponds to "I do not agree at all" and 0 to to "I totally agree".

The demographic characteristics of the sample are the following: All 56 participants are students (undergraduate or postgraduate/PhD) at the University of Rostock. Most of them were recruited during lectures or seminars they attended. This ensured that we had participants from various departments of the university. We classified the subjects they study according to four groups. 14.3 % study Arts and Humanities, 51.8 % Science and Technology, 5.4 % Health and Life Sciences and 28.6 % Social Sciences (see Figure 6.6a). 54.6 % are male, 46.4 % are female. Technophilia values ranged from 2 to 4 among participants: 14.3 % had a value of 2, 46.4 % had 3, and 39.3 % had 4 (see Figure 6.6b).



(a) The distribution of participants across subject groups.



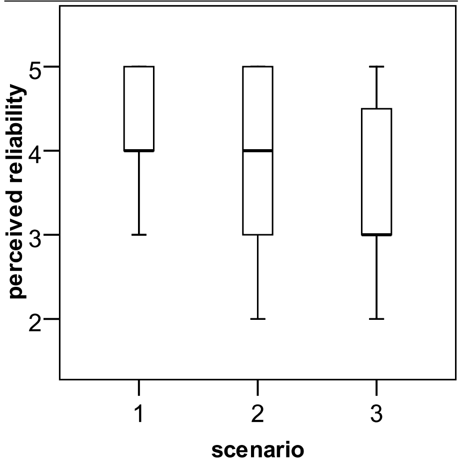
(b) The distribution of technophilia values across participants.

Figure 6.6: Characteristics of the sample.

After filling out the first questionnaire, each subject took part in the three trials (see Figure 6.4). The sequence of the trials was varied among subjects to avoid order effects. If the subject was in the training group, s/he was trained prior to the first trial. During the experiments, each click in the user interface of the assistance system and each action of the automatic assistance was logged.

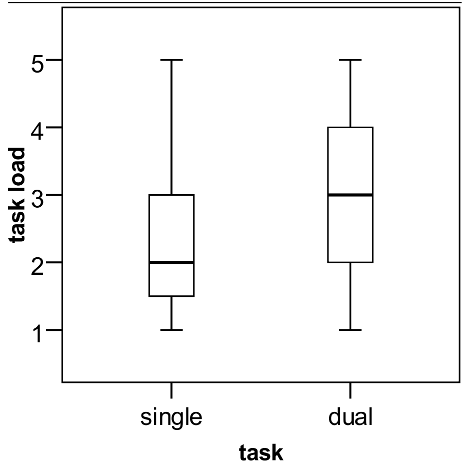
Before each trial, the subject read the scenario description. If the subject was in the dual task group s/he was given the arithmetic tasks to solve during the trial. The participant would then carry out the trial. Each trial was followed by a questionnaire to assess perceived usefulness (PU), perceived ease of use (PE), and the perceived reliability of the assistance system (manipulation check). All of those constructs were assessed using five-point Likert scales (0 to 4). The PU and PE values were calculated by summing up the six items of each scale. Thus, the range

		perceived reliability	
Scenario	N	μ	σ
1	56	4.20	0.80
2	56	4.18	0.88
3	56	3.63	0.98
overall	168	4.00	0.92
$F(2,165) = 7.49, p \leq .001$			
$f = 0.30$ (medium effect size)			



(a) Perceived reliability depends on the scenario.

		task load	
task	N	μ	σ
single	28	2.29	1.12
dual	28	2.96	1.17
overall	56	2.63	1.18
$t(54) = -2.22, p \leq .031$			
$d = 0.59$ (medium effect size)			



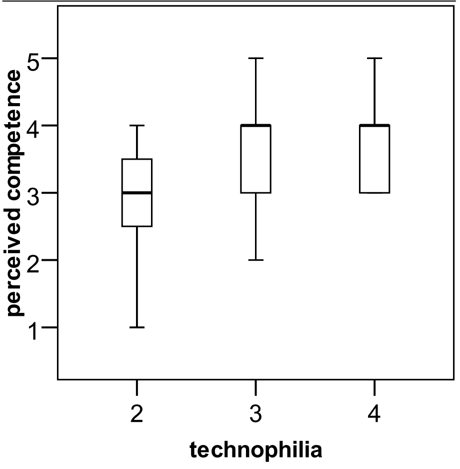
(b) The participants' perceived task load level depends on the number of tasks.

Figure 6.7: Influence of scenario on perceived reliability; influence of the number of tasks on participants' perceived task load level.

of both PU and PE was 0 to 24. If people switched to manual control during the trial, they were also asked for their reasons for switching. Possible answers were, *"The automatic configuration took too long"*, *"The automatic assistance carried out too many useless actions"*, and *"I thought I could solve the task better manually"*. There was also an option labeled *"Other"*, where people could write down any other reasons they had.

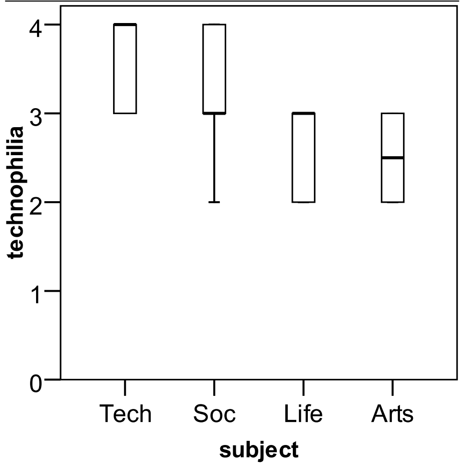
Having completed all three trials, subjects filled out another questionnaire containing the statements *"I experienced a certain task load during the trials"* (manipulation check to find out whether the dual task group felt a higher task load than the single task group) and *"I feel competent using the assistance system"* (manipulation check for the effects of training, i.e. experience), both to be rated on a five-point Likert scale. Furthermore, subjects were asked to rank order the three scenarios according to their perceived satisfaction with the assistance system. In addition, the questionnaire included the open question, *"Do you have any more*

		perceived competence	
technophilia	N	μ	σ
2	8	2.88	0.99
3	26	3.69	0.68
4	22	3.86	0.64
overall	56	3.64	0.77
$F(2,53) = 5.74, p \leq .006$			
$f = 0.47$ (large effect size)			



(a) The participants' perceived competence in operating the assistance system depends on technophilia.

		technophilia	
subject	N	μ	σ
Arts	8	2.50	0.54
Soc	16	3.25	0.78
Tech	29	3.52	0.51
Life	3	2.67	0.58
overall	56	3.25	0.69
$F(3,52) = 6.96, p \leq .001$			
$f = 0.63$ (large effect size)			



(b) The subject a participant studies is correlated with technophilia.

Figure 6.8: Influence of technophilia on perceived competence; relation between subject and technophilia.

comments?" We discuss the answers people gave in Section 6.4.3.

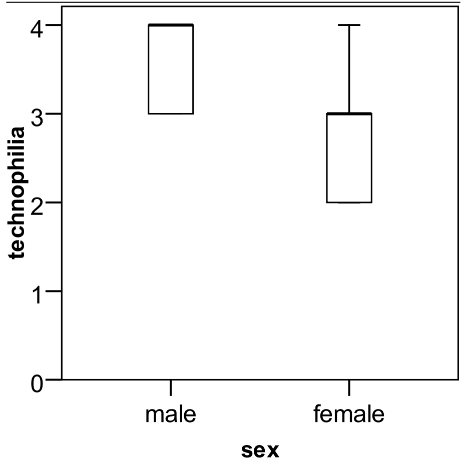
In the end, the participant received the money and was dismissed. In many sessions, a discussion about the assistance system developed at this point. The experimenter wrote down any interesting comments or suggestions given during such discussions immediately after the participant had left.

6.3.3 Hypotheses

The hypotheses we want to test in the study are the following:

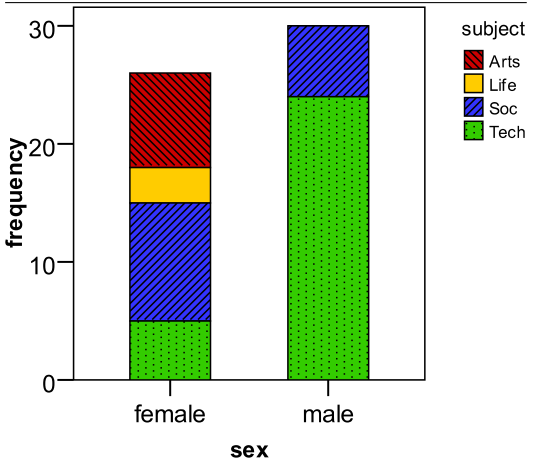
- Task load influences user acceptance of the assistance system.
- Experience influences user acceptance of the assistance system.
- The behavior of the automatic assistance influences user acceptance of the assistance system.

	technophilia		
sex	N	μ	σ
male	30	3.67	0.48
female	26	2.77	0.59
overall	56	3.25	0.69
$t(54) = 6.30, p \leq .001$			
d = 1.67 (large effect size)			



(a) Technophilia depends on participants' sex.

		subject			
sex	N	Arts	Life	Soc	Tech
male	30	0	0	6	24
female	26	8	3	10	5
$\chi^2(3, N = 56) = 24.29, p \leq .001$					
ϕ = 0.66 (large effect size)					



(b) The subject a person studies is correlated with her/his sex.

Figure 6.9: Influence of participants' sex on technophilia; relation between sex and subject.

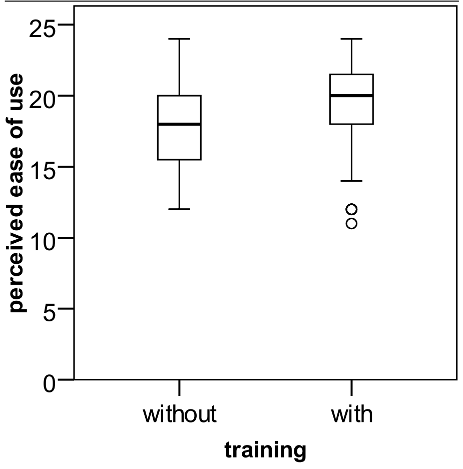
- Technophilia influences user acceptance of the assistance system.

6.4 Results of the User Study

In the following subsections, the results of the user study are presented in text, plots and tables. To check for statistical significance, we use two-sided t-tests (or Welch tests if the equal variances assumption is violated), analyses of variance (ANOVAs) and χ^2-tests. Furthermore, we use an α level of .05 for all statistical tests. In other words, a result is considered statistically significant if its p value is $< .05$. If a result has a p value of .05, there is a probability of 5 % that this result is due to chance. In the tables, we always specify the p value of the respective result. Consider for example Figure 6.7a. Here, $p \leq .001$, which reads as "The probability that perceived reliability does not depend on the scenario is at most 0.001."

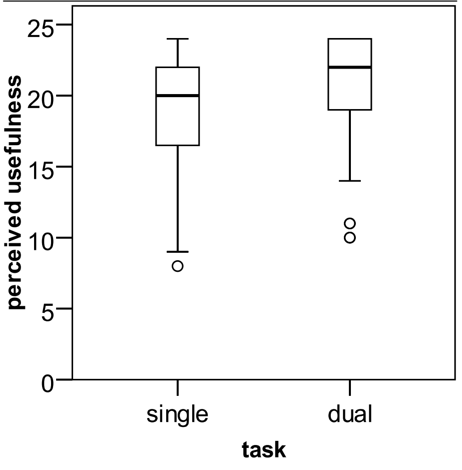
For decades, there has been a debate among researchers about the value of

		PE	
training	N	μ	σ
without	84	17.95	2.98
with	84	19.26	3.07
overall	168	18.61	3.09
$t(166) = -2.81, p \leq .006$			
$d = 0.43$ (medium effect size)			



(a) Perceived ease of use (PE) of the assistance system depends on whether a participant was trained or not.

		PU	
task	N	μ	σ
single	84	19.00	3.96
dual	84	20.80	3.63
overall	168	19.90	3.89
$t(166) = -3.07, p \leq .003$			
$d = 0.47$ (medium effect size)			



(b) Perceived usefulness (PU) depends on whether a participant has to solve a secondary task.

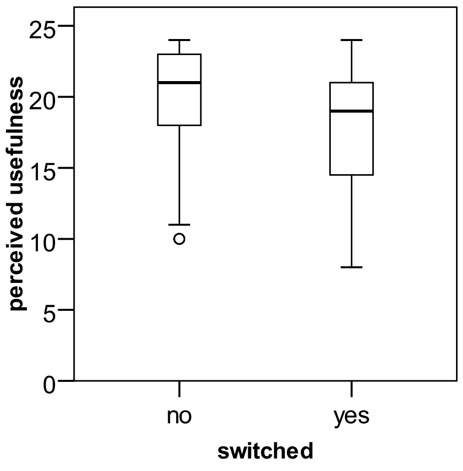
Figure 6.10: Influence of training on perceived ease of use (PE); influence of the number of tasks on perceived usefulness (PU).

statistical significance tests. Ziliak and McCloskey point out that these tests can cause false hypotheses to be accepted and correct hypotheses to be rejected and should therefore be abandoned [Ziliak & McCloskey 2004]. Another problem is that statistical significance does not say anything about the size of an effect, i.e. the strength of the relationship between two variables. Thus, a small effect may be statistically significant, even though it has little practical relevance. Furthermore, the widely used α level of .05 is quite arbitrary.

On the other hand, many researchers think statistical significance tests have their place [Frick 1996]. We, too, think they are useful if we keep in mind two things:

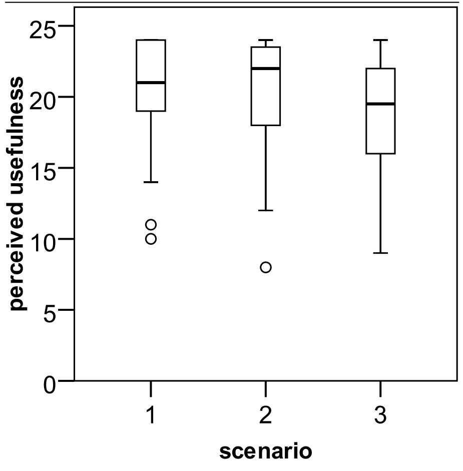
1. If we find that a result is not statistically significant, this does not mean there is no effect. Maybe there is one, but we were just not able to prove it in this study.

		PU	
switched	N	μ	σ
no	141	20.28	3.55
yes	27	17.93	4.95
overall	168	19.90	3.89
$t(31.33) = 2.36, p \leq .025$			
$d = 0.54$ (medium effect size)			



(a) PU is correlated with switching to manual configuration.

		PU	
Scenario	N	μ	σ
1	56	20.43	3.53
2	56	20.59	3.50
3	56	18.68	4.35
overall	168	19.90	3.89
$F(2, 165) = 4.32, p \leq .015$			
$f = 0.23$ (medium effect size)			



(b) PU depends on the scenario.

Figure 6.11: Relation between PU and switching to manual configuration; influence of scenario on PU.

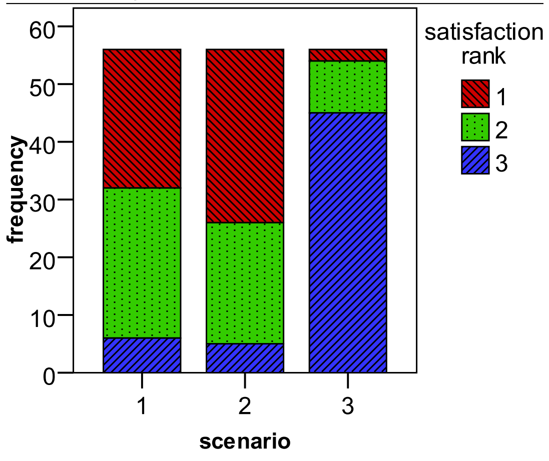
2. Even if we find a statistically significant result, there is a chance (in our case, at most 5 %) that there is no effect and we obtained this result by chance.

To assess whether an effect is large enough to be practically relevant, we consider the effect size in addition to statistical significance. Different effect size measures exist for different statistical tests. We use Cohen's d for the t-tests, Cramer's ϕ for the χ^2-tests and Cohen's f for the ANOVAs [Bortz & Döring 2006]. Table 6.3 lists which values correspond to a small, medium, and large effect for those three measures.

	classification of effect sizes		
effect size	small	medium	large
Cohen's d	0.20	0.50	0.80
Cramer's ϕ	0.10	0.30	0.50
Cohen's f	0.10	0.25	0.40

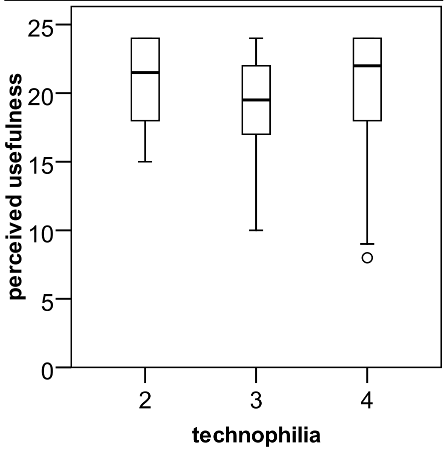
Table 6.3: Classification of effect sizes.

		rank		
Scenario	N	1	2	3
1	56	24	26	6
2	56	30	21	5
3	56	2	9	45
$\chi^2(4, N = 168) = 87.21, p \leq .001$				
$\phi = 0.51$ (large effect size)				



(a) People were most satisfied in Scenario 2 and least in Scenario 3.

		PU	
technophilia	N	μ	σ
2	24	20.67	3.13
3	78	19.05	3.66
4	66	20.62	4.23
overall	168	19.90	3.89
$F(2, 165) = 3.56, p \leq .031$			
$f = 0.21$ (medium effect size)			



(b) PU depends on technophilia.

Figure 6.12: Influence of scenario on satisfaction; influence of technophilia on PU.

In Box-Whisker plots (see e.g. Figure 6.12b) the box represents the interquartile range, the horizontal line in the box represents the median, the ends of the whiskers represent the minimum/maximum, ° represents an outlier (between 1.5 times and 3 times the interquartile range above/below the quartile) and * represents an extreme value (more than 3 times the interquartile range above/below the quartile).

6.4.1 Manipulation Checks

An analysis of variance showed that system behavior had a significant influence on perceived reliability (see Figure 6.7a). The Scheffé post hoc test [Maxwell & Delaney 2003] revealed that participants found the system significantly more reliable in Scenarios 1 and 2 than in Scenario 3, but that there is no significant difference between Scenarios 1 and 2. This is because several people did not perceive the useless action of the system (opening the converted PDF document on the colleague's notebook) in Scenario 2 as disturbing because they did not even know that it was caused by the assistance system: One participant commented that many conversion tools open the converted document so he did not know whether to attribute this action to the assistance system or the conversion tool used. These results indicate that the manipulation with respect to system behavior was partially successful.

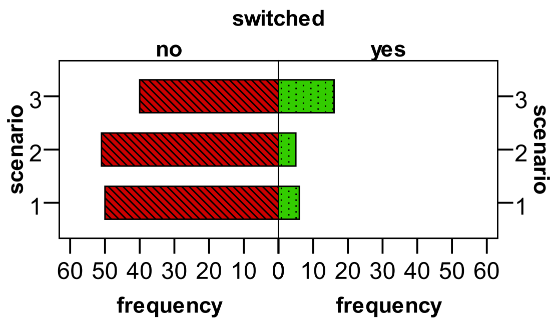
A two-tailed t-test showed that participants in the dual task group felt a significantly higher task load than those in the single task group (see Figure 6.7b). This indicates that the manipulation with respect to task load was successful.

A two-tailed t-test did not show that participants that were trained feel significantly more competent than those that were not trained, $t(54) = -1.40, p \leq .17$. This is probably due to the fact that perceived competence rather depends on a participant's level of technophilia than on training: An analysis of variance showed that the more technophilic a participant is, the more competent s/he feels operating the assistance system (see Figure 6.8a for details). However, perceived ease of use among participants that were trained was significantly higher than among participants that were not trained (see Figure 6.10a for details). This indicates that participants developed some routine after training. We can conclude that the manipulation with respect to training was successful.

6.4.2 Quantitative Findings of the User Study

Demographic Data Before discussing the results concerning the assistance system, we present a few results regarding the demographic data of the sample. These results are not directly relevant for answering our research questions, but may help the reader to get an understanding of the participants. An analysis of variance showed a relation between the subject a participant studies and technophilia (see Figure 6.8b for details). Social Sciences (Soc) and Science and Technology (Tech) students are more technophilic than Arts and Humanities (Arts) and Health and Life Sciences (Life) students. Furthermore, a t-test showed that technophilia depends strongly on the sex of a participant, with male participants being significantly more technophilic than female participants (see Figure 6.9a). Furthermore, a χ^2-test showed that the subject a person studies depends on her/his sex (see Figure 6.9b for details). In t-tests, we checked if technophilia values vary between the training/no training groups or the single/dual task groups. We found no significant

		switched	
Scenario	N	no	yes
1	56	50	6
2	56	51	5
3	56	40	16
overall	168	141	27
$\chi^2(2, N = 168) = 9.80, p \leq .007$			
$\phi = 0.24$ (medium effect size)			



(a) Whether or not participants switched to manual configuration depended on the scenario.

Figure 6.13: Influence of scenario on switching to manual control.

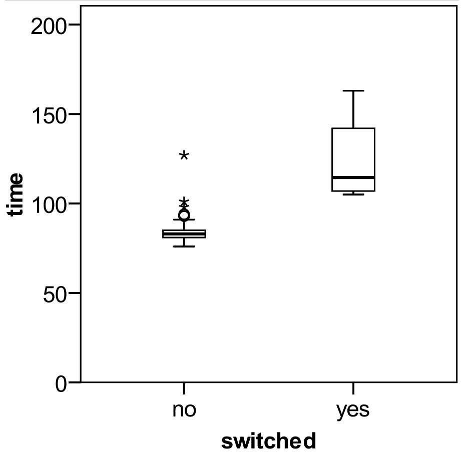
difference. Thus, we can conclude that subjects with similar technophilia values were not accidentally assigned to the same groups.

Perceived Usefulness and Perceived Ease of Use In the following, we present the results regarding user acceptance of the assistance system. As already mentioned in Section 6.4.1, experience using the system influences PE: People that were trained perceived the system as significantly easier to use than those that were not (see Figure 6.10a). Apparently, people developed some routine after training. This indicates they proceeded from Level 1 to Level 2.

PU depends on a number of factors. A two-tailed t-test showed that task load had a significant effect on PU: Subjects from the dual task group rated the assistance system higher in terms of PU than subjects from the single task group (see Figure 6.10b). We can conclude that people that experience a higher level of task load due to some secondary task value automatic assistance more than people with a low task load. One explanation is that the assistance system relieves people of cognitive load, giving them more time to attend to their secondary task.

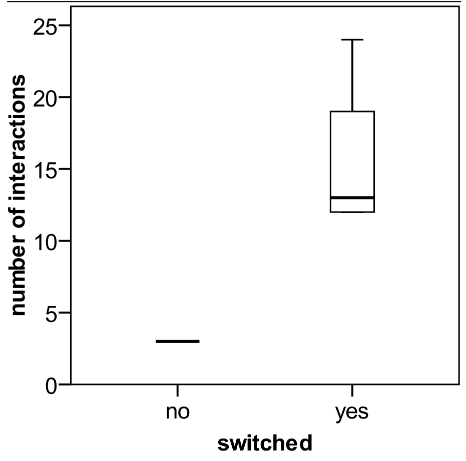
A two-tailed t-test showed that switching from automatic to manual configuration is correlated with a lower PU value (see Figure 6.11a). As both are dependent

		time in seconds	
switched	N	μ	σ
no	50	85.08	7.89
yes	6	124.33	23.15
overall	56	89.29	15.95
$t(5.14) = -4.13, p \leq .009$			
$d = 2.27$ (large effect size)			



(a) Switching to manual control had a strong influence on the time taken to configure the room – Scenario 1.

		number of interactions	
switched	N	μ	σ
no	50	3.00	0.00
yes	6	15.50	4.97
overall	56	4.34	4.18
$t(5.00) = -6.16, p \leq .002$			
$d = 3.56$ (large effect size)			



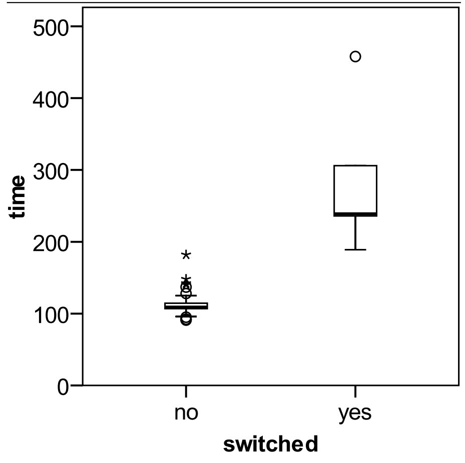
(b) Switching to manual control influenced the number of interactions with the assistance system – Scenario 1.

Figure 6.14: Scenario 1: Influence of switching to manual control on time and number of interactions.

variables, it is not clear whether people switch because they perceive the system as not useful enough or vice versa. However, PU is notably high even among subjects that switched ($\mu = 17.93$). Comments given by participants explain this finding: Several people said they preferred a system that occasionally fails to pure manual control because in most cases the automatic assistance system works fine and then it relieves them of work and saves time. This result is consistent with the findings of Parasuraman and Riley [Parasuraman 1997] and Muir and Moray [Muir & Moray 1996] that occasional system failures do not deter people from using the system in the future. Apparently, people accept assistance systems even if they are imperfect (i.e. exhibit Level 2 behavior).

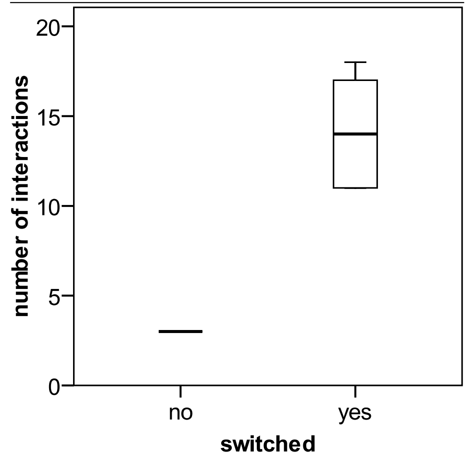
An analysis of variance showed that PU depends on system behavior. PU is lower for Scenario 3 than for Scenarios 1 and 2 (see Figure 6.11b). Furthermore, a χ^2-test confirmed that there is a relation between system behavior and switching to manual mode: People switched more often in Scenario 3 than in Scenarios 1 and

		time in seconds	
switched	N	μ	σ
no	51	112.75	15.57
yes	5	285.60	105.00
overall	56	128.18	59.13
$t(4.02) = -3.68, p \leq .021$			
$d = 2.30$ (large effect size)			



(a) Switching to manual control had a strong influence on the time taken to configure the room – Scenario 2.

		number of interactions	
switched	N	μ	σ
no	51	3.00	0.00
yes	5	14.20	3.27
overall	56	4.00	3.34
$t(4.00) = -7.66, p \leq .002$			
$d = 4.84$ (large effect size)			



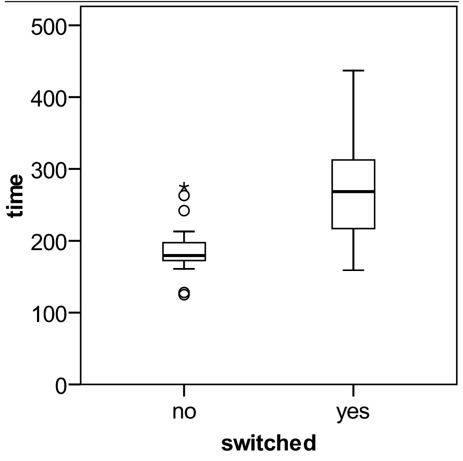
(b) Switching to manual control influenced the number of interactions with the assistance system – Scenario 2.

Figure 6.15: Scenario 2: Influence of switching to manual control on time and number of interactions.

2 where the action sequence produced by the assistance system was more optimal (see Figure 6.13a). Furthermore, this finding corresponds with another result: One question in the final questionnaire asked people to rank the scenarios according to satisfaction (ties were not allowed). A χ^2-test showed that the results are significant (see Figure 6.12a). It surprised us a little that Scenario 2 was ranked higher than Scenario 1, although the system took longer to find a solution and produced one useless action. Comments given by users after the trials suggest two reasons: First, several people did not perceive the useless action as a useless action. Thus, they thought it was Level 1 behavior when it was actually Level 2 behavior. Second, people felt that the benefit was higher in Scenario 2 because the automatic assistance saved more configuration work than in Scenario 1.

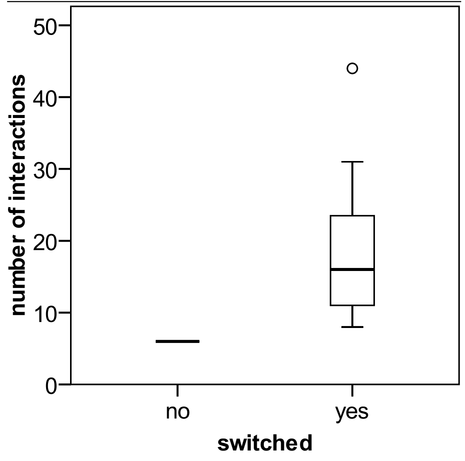
An analysis of variance also showed that PU depends on one of the control variables – how technophilic a person is. Surprisingly, participants with a technophilia value of 2 or 4 rated the assistance system more useful than those with 3 (see Fig-

		time in seconds	
switched	N	μ	σ
no	40	185.73	28.56
yes	16	275.25	83.40
overall	56	211.30	64.35
$t(16.43) = -4.20, p \leq .001$			
$d = 1.44$ (large effect size)			



(a) Switching to manual control had a strong influence on the time taken to configure the room – Scenario 3.

		number of interactions	
switched	N	μ	σ
no	40	6.00	0.00
yes	16	18.19	9.67
overall	56	9.48	7.51
$t(15.00) = -5.04, p \leq .000$			
$d = 1.78$ (large effect size)			



(b) Switching to manual control influenced the number of interactions with the assistance system – Scenario 3.

Figure 6.16: Scenario 3: Influence of switching to manual control on time and number of interactions.

ure 6.12b). A possible reason is that less technology-savvy people are glad because the automatic assistance system relieves them of cumbersome configuration tasks and very technophilic people like it because of a certain “coolness” factor, while averagely technophilic people are happy with automatic or manual configuration, so it does not make such a big difference to them.

User Performance Although our emphasis is on user acceptance, we now present some interesting results regarding user performance. Figures 6.14a, 6.15a, and 6.16a show how much time participants took for their tasks in Scenarios 1, 2, and 3. The t-tests reveal that people that switched to manual control took significantly more time than those that did not switch. Figures 6.14b, 6.15b, and 6.16b show that they also had significantly more interactions with the assistance system in each scenario. These are interesting results. Several people commented they switched to manual control because they felt the automatic assistance was slow and they could configure the room quicker when doing it manually. The figures show that this was not the case. However, some participants' comments indicate that it gave them greater satisfaction to switch to manual control than to wait for the automatic assistance to finish.

No Interaction Effects To conclude this section, one thing remains to be said: We have reported only those results that are statistically significant at the .05 level and omitted those that failed to meet this criterion. Specifically, we could not prove any significant interaction effects of the factors we considered in the study. The characteristic feature of an interaction is that the effect of one factor on a variable depends on the value of another factor [Bortz & Döring 2006]. For example, in our study, no significant interaction could be found between the factors task load and experience with respect to PU.

6.4.3 Comments Given by the Participants

45 subjects answered the open question "Do you have any more comments?" and gave a total of 87 comments. Using these comments, we performed a content analysis according to Mayring [Mayring 2000]: We first analyzed the comments to form categories inductively and then grouped the comments according to these categories. In the following, the categories are presented. To give the reader some ideas about the participants' actual comments, we provide examples in each category. We believe these qualitative results are just as important as the quantitative results as they provide a feeling of what participants particularly liked and disliked about the assistance system. Thus, they can help to interpret the quantitative findings and to identify future research avenues.

Time matters Nine subjects commented that they found the automatic assistance took long to fulfill the goals. For example, one participant wrote, "The assistance system is useful for starting a presentation, but slow in deciding what to do." However, two participants stated that although they found the automatic configuration a bit slow, they did not think manual configuration would have been quicker. This corresponds with our findings (see Figures 6.14a, 6.15a, and 6.16a).

People think automatic assistance saves time and work Fourteen subjects said the assistance system saved them time or work or both. One participant wrote, “The assistance system is useful and saves a lot of time”. Another said, “The assistance system reduces workload tremendously”. Several participants mentioned the assistance system was especially suited for non-expert computer users. For example, one participant stated “Especially for less competent users the tool makes giving a talk a lot easier. It relieves the presenter of the whole stress of manually configuring the devices. However, if anything fails, it is easy to intervene and perform configuration manually.”

People think the assistance system is easy to use Five people wrote they thought the assistance system was easy to use, e.g. because it is self-explanatory and easy to learn. For example, one subject said, “I am not that skilled when trying new (technical) things, but I understood everything instantly and found the system convenient to use”. Nobody found the assistance system cumbersome to use. These comments suggest that the overall perceived ease of use value of 18.61 can be interpreted as being quite high.

People accept even imperfect assistance Nine subjects commented that they would use the automatic assistance if it does the right thing most of the time. They said it was no big problem if the system occasionally failed, provided there was a possibility to intervene and perform the configuration manually. With some experience they would know when it is time to switch to manual control because the automatic assistance is likely to fail. People argued that a system that fails sometimes is better than having to perform all the configuration manually. This finding is consistent with results of Muir and Moray that occasional system failure does not cause people to lose trust in a system [Muir & Moray 1996].

People prefer the automatic mode Five participants wrote that they prefer the automatic mode to the manual mode. One participant stated he generally preferred manual configuration of electronic devices and gave the hi-fi system at his home as an example. This results in 5:1 in favor of the automatic mode. Of course, this finding should be taken with a grain of salt as it covers only the small percentage of people that explicitly stated which mode they prefer. However, it is consistent with other ideas previously identified, e.g. with the idea that automatic assistance saves time and work.

People can get irritated by the automatic assistance Four people commented that they found some actions of the automatic assistance confusing. This applies in particular to Scenario 3. For example, one participant wrote, “If the system tries out several actions it might irritate a user”. Irritating actions are an important issue. As it is not possible to prevent them completely, research must address the question how to enable users to cope with them.

System feedback must be improved Several subjects said the fact that the automatic assistance kept informing them about what it was doing was extremely important and they would not have accepted it without this feedback. However, only one participant stated that s/he was satisfied with the feedback the system gave, while ten subjects said that this feedback could be improved. For example, one participant said, “More pictures or small icons to visualize actions of the system would be good”. Another requested to “present actions of the system more clearly”.

People want to be able to explicitly specify goals Many people said they liked the concept of goal-based interaction: It puts the focus on *what* should be accomplished rather than *how* to accomplish it. However, it struck us as a surprise that many people were not very enthusiastic about the concept that the assistance system infers their goals and automatically tries to fulfill them. We identified four different objections people have against this concept: The major concern the participants uttered was that they did not believe the intention analysis always adequately infers their goals. Several people criticized that there is no possibility to manually correct incorrectly inferred goals if the intention analysis fails. Some people also commented they would like more interaction with the system. They would feel less passive and more in control if they could explicitly enter their goals into the assistance system. Some participants said they felt patronized by the assistance system because it claimed to know their goals. People also thought the system would be much more flexible if it gave them the possibility to explicitly enter goals. Participants uttered two different options for improvement. Nine people said they would prefer a user interface for entering their goals to the intention analysis. Two participants said the intention analysis should present suggestions for goals, but should include the possibility to alter incorrect goals via a user interface.

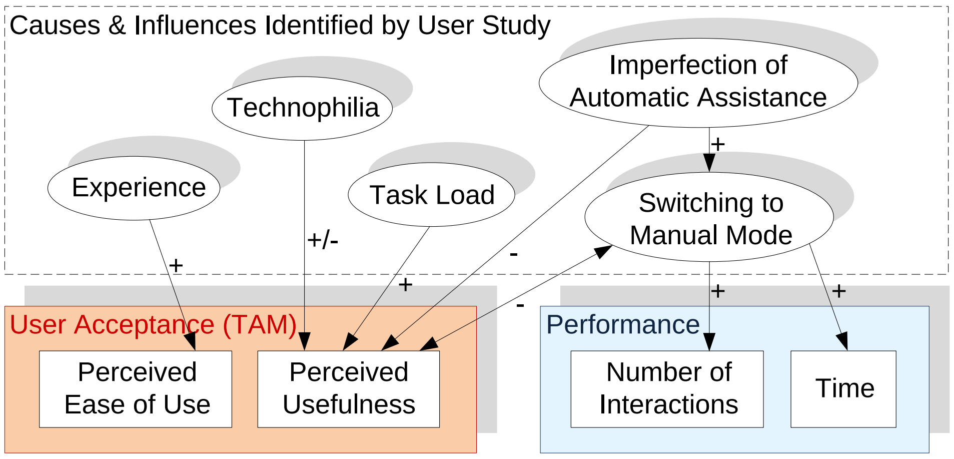


Figure 6.17: The proposed scheme of user acceptance and performance.

6.5 Scheme of User Acceptance and Performance

Using the results of the user study, we develop a unified scheme of user acceptance and performance for Ubiquitous Computing assistance systems based on Davis' Technology Acceptance Model [Davis 1989] (see Figure 6.17). The scheme shows which factors directly or indirectly influence PU and PE (determinants of user acceptance) as well as the number of user interactions with the assistance system and the time taken to fulfill the user's goals (determinants of user performance). All arrows in the scheme correspond to influences identified in the study. Each arrow points from the determining factor to the variable influenced by that factor, with the exception of the two-headed arrow between PU and switching to manual control: Due to the fact that both are dependent variables, we cannot say whether PU determines if people switch to manual control or vice versa. A "+" sign next to an arrow marks a positive correlation, while a "–" sign marks a negative correlation.

As an example, let us look at the leftmost arrow. It should be read in the following way: If a person has experience using the system, s/he is likely to perceive it as easier to use than a person with no experience.

One relation needs further explanation: The correlation between technophilia and PU is not linear. Thus, there is a "+/−" sign next to the arrow. All other relations should be easy to read as this scheme is in fact a summary of the results presented above.

This scheme is necessarily incomplete as it comprises only those factors found to be relevant in this study. However, we believe that it can help to understand what influences user acceptance not only of this system, but of other proactive assistance systems (e.g., those introduced in Section 6.1) as well. Furthermore, it can serve as a basis for discussion among researchers wishing to further investigate the notion of user acceptance of Ubiquitous Computing applications.

6.6 Ad-hoc Assistance Systems and Wandke's Framework

Based on the results of our study, we can now discuss the question which kind of ad-hoc assistance system in terms of Wandke's assistance framework is acceptable to the users of Ubiquitous Computing applications.

- **The stages of human-machine interactions that can be assisted**: As the system used for the study supports only the *decision making/action selection* stage of human-machine interactions, we can only comment on which kind of assistance is appropriate at this stage. Recall that our hypothesis was that *informative execution assistance* is acceptable. In general, our results have shown that this is the case. The system used in this study is an informative execution assistance system, and user acceptance was fairly high. However, our results also show that this is a simplification. Acceptance of Ubiquitous Computing applications is context-dependent and varies according to the parameters task load, experience, system behavior, and technophilia. This indicates that when users experience a certain task load, users are experienced, the system finds a good solution quickly, or people have a technophilia value of 2 or 4, they accept an informative execution assistance system. However, if those conditions are not met, they might prefer another level that gives them more control, such as *take-over assistance* or *delegation assistance*. Thus, our hypothesis that *informative execution assistance* is acceptable for users of ad-hoc assistance system was only partially correct: Future ad-hoc assistance systems should be capable of different levels for different contexts.

- **Adjustment**: Our hypothesis was that fixed assistance is appropriate. The assistance system we used in the study provides *fixed assistance*. However, the discussion above has shown that ad-hoc assistance systems must be capable of different levels of assistance. Thus, our hypothesis regarding adjustment was not correct: Fixed assistance is not the best solution in ad-hoc environments. The question is whether the *user* should adapt the level according to her/his preferences, or if the system should adapt itself according to the context. The former corresponds to *adaptable assistance*, while the latter corresponds to *adaptive assistance*. We believe that in ad-hoc environments *adaptive assistance* is more suitable: In an unknown situation and/ or a situation that imposes a certain task load on the user, s/he probably does not have the time or cognitive resources to adapt the system her-/himself. In the next section, we give some hints how adaptive assistance can be realized.
- **Initiative**: The assistance system used in the study provides *active assistance*. A high number of participants did not switch from automatic to manual mode, and users' comments indicate that they are satisfied with the automatic assistance even if it occasionally fails. These results show that most people are satisfied with active assistance, provided that they can easily take over manually if anything goes wrong. This indicates that our hypothesis regarding initiative was correct.

6.7 Implications for User Interface Design of Ubiquitous Computing Systems

As discussed in the previous section, *adaptive assistance* is preferable over *fixed assistance* in ad-hoc environments. A good starting point for making an assistance system adaptive according to the current context is to make the user interface adaptive. Research on context-aware user interfaces has focused mainly on questions such as how to adapt the user interface layout to output devices with different properties, e.g. screen size or resolution [Butter *et al.* 2007], or the type of output device (head-mounted display, handheld device, etc. [Witt *et al.* 2007]). Liu et al. propose a user interface that adapts according to patterns it has recognized in the user's interaction behavior [Liu *et al.* 2003]. While this is a good starting point, the results of the study suggest that for the kind of application studied here, more factors should be taken into account. We propose to broaden the notion of *context*: Whether the user is technophilic or not, whether s/he is experienced or a novice, whether s/he experiences a certain task load, and the behavior of the system itself – whether it is able to fulfill her/his goals in an optimal way or is likely to produce errors – as our study revealed, these parameters influence the user's information and interaction preferences with the system. We believe that this should be reflected in the user interface. To be more precise, the user interface should be adapted according to context in this broader sense. And by *adapt* we do not just mean the appearance of single user interface elements, but the whole structure of the user interface, including how many and which kinds of elements it contains. Concerning the experienced vs. novice dichotomy, the need for adaptivity has been recognized even in desktop computing: Many programs contain wizards that guide novice users in performing the most common tasks, while expert users work with a more advanced interface. Concerning the other parameters, we give some hints which kind of adaptation is worth to be considered.

One possibility for adaptation has been suggested by the participants themselves: Some wanted to abandon the intention analysis in favor of the possibility to enter their goals into the system themselves, while others requested the possibility to correct goals that were incorrectly inferred by the intention analysis. We feel that these preferences might be related to a person's technophilia value. However, due to the small number of people who gave these comments we have too little evidence to determine whether this is indeed the case. Our study has also shown that people experiencing a certain task load (due to a secondary task) want to be able to grasp the current system state at a short glance. Furthermore, they do not want to be alarmed immediately if the automatic assistance is likely to produce a suboptimal solution. This is because they are occupied elsewhere and are thus likely to accept suboptimal solutions. In contrast, relaxed people are more likely to closely monitor the system and prefer more detailed information about the system state. They need an indicator that shows how likely the system will find a good solution quickly. This enables them to decide whether and when to intervene manually. Of course, these ideas raise many issues. How to capture the required context adequately and how to transfer it into appropriate user interface elements at run-time are just two of them. This deserves further investigation.

6.8 Limitations of the Study

Five limitations of our study need to be acknowledged:

- We used the PU and PE scales from the original version of TAM to assess user acceptance. TAM was developed for desktop applications and has its shortcomings when applied to Ubiquitous Computing. For example, it includes neither social acceptability nor trust, concepts deemed important in Ubiquitous Computing [Connelly 2007]. However, it allows to measure perceived usefulness and perceived ease of use with a small number of items.
- Our results do not apply to all Ubiquitous Computing systems. We only measured people's acceptance of one exemplary system. Nevertheless, we believe that our results generalize to systems of the kind discussed in Section 6.1. Those systems and our system have in common that they a) proactively assist the user and b) perform run-time strategy synthesis based on declarative descriptions of user goals, which can yield suboptimal solutions. On the other hand, we believe that our results – to some extent – generalize to different applications that people use under the influence of similar factors as in our study. Consider, for example, driving a car. Here, too, users have more than one task at a time. The primary task is driving the car. The secondary task is planning which route to take. People occasionally leave this task to a navigation system, especially in areas they do not know well. The results of our study suggest that here people tend to accept suboptimal routes.
- The characteristics of the sample were not optimal. All subjects were university students that were moderately to very technophilic. Thus, we cannot say whether our findings generalize to other demographic groups.
- We did not perform a longitudinal study and can thus not give any indication how acceptance develops over a longer period of use (Phase 3).
- The fifth limitation concerns a problem common to all lab studies: Influences like ethical questions or people's motivation are generally hard to assess in a lab study because a lab is necessarily to some extent an artificial environment. As an example, consider the question how the dual task group would

have acquitted themselves if they had lost their own money. Supposedly their motivation to finish the trials quickly would have been higher. However, in this case, we would probably not have been able to recruit enough participants for the study.

6.9 Chapter Summary

In this chapter, we have presented a user study that investigates whether users accept an assistance system based on the AdDCo algorithm. This algorithm does not completely fulfill the requirement *rationality*. Hence, *system behavior* was one factor in the study. The other two factors *experience* and *task load* refer to the context of use. Participants had the task of configuring a room equipped with a variety of electronic devices for a meeting. The assistance system helped them with this task. It consisted of an automatic and a manual mode. The automatic mode was the default, but participants could switch to manual control if they wanted to.

In summary, one can say that user acceptance was fairly high across all experimental conditions, even when the assistance system produced suboptimal solutions. We can thus conclude that an assistance system need not completely fulfill the requirement *rationality* in order to be accepted by its users, provided that it offers the users enough benefit. This is the most important result of the study. It refutes the assumption that users will not accept an assistance systems if it exhibits suboptimal behavior, which we discussed in Section 6.2.

A second result of the study is that user acceptance is significantly influenced by:

- task load: Under the influence of an increased task load, people perceived the automatic assistance as more useful than when relaxed, probably because it relieved them of workload.
- experience: People that had some experience with the assistance system (Phase 2) found the assistance system easier to use than those with no experience (Phase 1).
- the behavior of the automatic assistance: When experiencing Level 2 behavior, people perceived the system as less useful and were more likely to switch to manual control than for Level 1 behavior. On the other hand, the more benefit the automatic assistance offered over pure manual control, the

more useful it was perceived and the more likely would people stick to the automatic assistance even if it exhibited Level 2 behavior.

- technophilia: Moderately and very technophilic people perceived the assistance system as more useful than averagely technophilic people.

We furthermore observed that people who switched to manual control took more time and had more system interactions than those that used the automatic assistance. Based on these results, we developed a unified scheme of user acceptance and performance for Ubiquitous Computing assistance systems. We could furthermore draw some conclusions which kind of assistance in terms of Wandke's framework is preferable in smart ad-hoc environments: *Informative execution assistance* is not always the best way to assist people in the *decision making/action selection* stage. According to the context, *take-over assistance* or *delegation assistance* may be preferable. Concerning *adjustment*, *adaptive assistance* is preferable over *fixed assistance*. Concerning *initiative*, active assistance is preferable over *passive assistance*. Our findings led us to discuss the need to make user interfaces of Ubiquitous Computing assistance systems context-aware in the broad sense outlined in Section 6.7. Based on the observations during the study, we gave a few ideas how this might be achieved.

CHAPTER 7

Providing Declarative Action Descriptions

Contents

As has become clear in the previous chapters, the basic building blocks of our strategy synthesis are declarative descriptions of the devices' possible actions. An important question that remains to be answered is how – i.e. *in which form* and *by whom* – such descriptions should be provided. This question is key for all strategy synthesis mechanisms that require action descriptions – the AdDCo algorithm as well as e.g. EMBASSI [Heider & Kirste 2002] or D-HTN [Amigoni *et al.* 2005].

Semantic web languages have been developed to enable machines to "understand" the content of the internet. This chapter investigates how to bring both worlds together. We show that semantic web services can in principle provide an appropriate formalism for generating action descriptions because they have several advantages over special-purpose languages such as PDDL [Ghallab *et al.* 1998]. This chapter serves two purposes: On the one hand, we strive to investigate the suitability of semantic web services for a whole class of approaches, including those named above. On the other hand, we are particularly interested to find out whether semantic web languages can be used with the AdDCo algorithm. Where suitable and possible, we therefore make some remarks that specifically concern the AdDCo algorithm.

This chapter is structured as follows: In Section 7.1 we briefly introduce previous attempts at using semantic web services with strategy synthesis mechanisms. In Section 7.2 we explain why semantic web languages are useful for smart environments and how they can be employed to generate the action descriptions we need for the strategy synthesis. We furthermore evaluate the benefits and shortcomings of existing semantic web languages with respect to our field of interest. Another important issue is who should provide these declarative descriptions. We suggest that large communities of stakeholders should generate and refine them collaboratively. This is explained in Section 7.3.

7.1 Related Work

Combining semantic web services with planning or similar techniques for strategy synthesis has been a topic of research in the past few years. One such project is OWLS-Xplan [Gerber *et al.* 2005] in the medical health domain. It combines Xplan, an AI planner based on Graphplan and HTN planning, with OWL-S by translating OWL-S service descriptions into PDDXML operators for planning. PDDXML is an XML dialect of PDDL. Unfortunately, Xplan is a centralized offline planner and can therefore not be used in dynamic environments. Wu et al. composed DAML-S (a predecessor of OWL-S) services [Wu *et al.* 2003] as well as OWL-S services [Sirin *et al.* 2004] with the HTN planner SHOP2. A similar approach is that of Qiu et al. [Qiu *et al.* 2006]. Sheshagiri et al. used a simple backward-chaining mechanism to compose OWL-S services [Sheshagiri *et al.* 2004]. Chen and Yang performed workflow planning using OWL-S service descriptions [Chen & Yang 2005]. All of these approaches rely on centralized architectures. To our knowledge semantic web services have not been used in combination with any decentralized approach yet.

7.2 Semantic Web Services

Generating the descriptions for the actions of devices in smart environments – which we have also called *operators* or *operator schemes* in the previous chapters – is a major challenge. At the moment they must be hand-coded for every device as devices are not equipped with such action descriptions by their vendors. Moreover, a widely accepted standard for describing the actions has not yet emerged. To make sure that operators of different origin are compatible, they must comply to a common terminology. For instance, a fixed set of predicates has to be defined. This could be done by means of common ontologies that all who write such operators agree upon. Approaches like the AdDCo algorithm will only have a chance to gain wide acceptance if those challenges are met. A special-purpose language like PDDL is well suited for describing actions, yet it provides no means to define the elements of such a description (e.g. predicates) using ontologies. Semantic web languages, on the other hand, allow to describe device actions with the help of ontologies. The key question is if existing semantic web languages qualify for describing the operators needed for our approach.

There are several languages that allow to either formulate ontologies or semantic web services or both. In the next sections, we evaluate how well the following languages are suited for describing device actions:

- OWL-S (Web Ontology Language for Services)
- WSMO (Web Service Modeling Ontology)
- DSD (DIANE Service Description)
- WSDL-S (Web Service Semantics)
- SAWSDL (Semantic Annotations for WSDL)

SAWSDL is based on WSDL-S, which itself is just a proposal. Nevertheless both approaches differ in some interesting points, so they are both worth investigating. We are especially interested in the possibilities of semantic annotation of a service. Especially, we need the ability to express preconditions and effects as they are the base for our approach as well as other approaches such as EMBASSI [Heider & Kirste 2002]. In the AdDCo algorithm, preconditions and effects are function-free first order literals. To be able to express that an effect holds for a certain set of devices, we also need universally quantified effects (see Section 5.1 for details). Hence, if a semantic web language is to be used with the AdDCo

	OWL-S [Martin *et al.* 2004]	**WSMO** [Lausen *et al.* 2005]	**DSD** [Klein 2004]	**WSDL-S** [Akkiraju *et al.* 2005]	**SAWSDL** [Kopecký *et al.* 2007]
semantic annotation of services	resources (apart from preconditions/effects) are named with URIs	all resources are URIs (including preconditions and effects)	all resources are URIs (including preconditions and effects)	supported, but no format specified	supported, but no format specified
preconditions and effects	string literals (must be processed by a separate parser)	included in WSMO standard	included, but preconditions are not expressive enough	supported, but no format specified	not supported
service grounding	not specified mapping needed	not specified mapping needed	included in standard	WSDL	WSDL
tool support	OWL-S editor as a plugin for Protégé [Elenius *et al.* 2005]	WSMO Studio (set of Eclipse plugins, quite complex) [Dimitrov *et al.* 2007]	templates for Microsoft Visio, not publicly available	Radiant annotation tool [Gomadam *et al.* 2005]	Radiant annotation tool [Gomadam *et al.* 2005]

Table 7.1: Comparison of different semantic web languages.

algorithm, it should support these constructs.

Additionally, the way the descriptions are grounded to a particular service is of interest. A service grounding describes the linking between the syntactical and the semantical description of a service. Finally, we take a look at the available tools for editing service descriptions. Table 7.1 summarizes the main findings of our comparison.

7.2.1 OWL-S (Web Ontology Language for Services)

OWL-S [Martin *et al.* 2004] services consist of three parts: the *service profile* that describes the functionality of the service and is used for service advertising and discovery, the *process model* which gives a more detailed description of the service, and the *service grounding* which specifies how to interact with the service. For our purposes, the profile and grounding are most relevant. The profile is the part of the OWL-S description that can accommodate our action descriptions and the grounding specifies how the service communicates with the outside world.

We now discuss how action descriptions can be expressed in OWL-S. OWL-S can be mapped to PDDL, as shown in [Gerber *et al.* 2005]. Mapping PDDL to OWL-S is also possible since OWL-S web service profiles include the properties *hasPrecondition* and *hasEffect*. These can accomodate PDDL expressions as string literals. Here is a ShowDoc operator in PDDL:

```
(:action ShowDoc
  :parameters (?Doc - Document ?Canv - Canvas)
  :precondition (and (SentToDisp ?Doc Projector1)
                     (CanvasDown ?Canv))
  :effect (and (DocShown ?Doc ?Canv)
               (forall (?OtherDoc - Document)
                  (when (not (= ?OtherDoc ?Doc))
                        (not (DocShown ?OtherDoc ?Canv))))))
```

The same operator is depicted as an OWL-S description in Figure 7.1.

In Section 5.1, we have argued that the notation we developed for the AdDCo algorithm is very similar to PDDL, but less expressive. This implies that OWL-S services can accomodate action descriptions for the AdDCo algorithm, provided

```
<process:AtomicProcess rdf:ID="ShowDoc">
        <process:hasPrecondition rdf:resource="#precondition"/>
        <process:hasEffect rdf:resource="#effect"/>
</process:AtomicProcess>

<process:Precondition rdf:ID="precondition">
       <expr:PDDL-Expression>
               <expr:expressionBody>
                      (and (SentToDisp ?Doc Projector1) (CanvasDown ?Canv))
               </expr:expressionBody>
       </expr:PDDL-Expression>
</process:Precondition>

<process:Effect rdf:ID="effect">
        <expr:PDDL-Expression>
                <expr:expressionBody>
                         (and (DocShown ?Doc ?Canv)
                                (forall (?OtherDoc)
                                         (when (not (= ?Doc ?OtherDoc))
                                                 (not (DocShown ?OtherDoc ?Canv))))))
               </expr:expressionBody>
        </expr:PDDL-Expression>
</process:Effect>
```

Figure 7.1: Mapping of a PDDL operator to OWL-S.

that the PDDL expressions in the *hasPrecondition* and *hasEffect* properties are restricted to the PDDL subset that can be understood by the AdDCo algorithm. Hence, preconditions should be conjunctions of function-free first-order literals. For effects, universal quantification is also allowed (see Section 5.1 for details).

Including PDDL expressions as string literals in OWL-S descriptions may seem reasonable at a first glance. However, there is a huge disadvantage: Only the syntax, but not the semantics of the PDDL expressions is mapped to OWL-S. To ensure interoperability of different web services, the semantics have to be defined externally as this is not included in the OWL-S standard. Furthermore, an extra parser is needed for the PDDL part of the service descriptions. Experiences in the EMBASSI project [Heider & Kirste 2002] have shown that this is a drawback. This disadvantage of OWL-S was also identified by the consortium that developed the OWL-S competitor WSMO [Fensel *et al.* 2006].

An OWL-S service is grounded by the process description. The inputs and outputs in the process description are usually defined as OWL classes. In contrast, the most common language for syntactic service descriptions is WSDL, wherein inputs and outputs are defined as XML schema data types. In order to reuse WSDL-based services, the OWL-S process description must be mapped to WSDL and vice versa. Due to its higher expressivity OWL classes are not compatible with primitive XML schema data types. Therefore a special mapping must be implemented for each ontology. A detailed overview of particular grounding approaches for OWL-S is given in [Kopecký *et al.* 2006].

OWL-S services can be edited with the OWL-S Editor, a plugin for Protégé [Elenius *et al.* 2005]. However, it is not very intuitive and development ceased more than four years ago.

7.2.2 WSMO (Web Service Modeling Ontology)

WSMO [Lausen *et al.* 2005] consists of four modelling elements: ontologies, goals, web services and mediators. We are especially interested in the web services description and in the definition of goals as we need a mechanism to express user goals. Similar to OWL-S, the WSMO standard does not specify how services should be grounded. This is a disadvantage as interoperability between services with different groundings may be hampered. [Kopecký *et al.* 2006] gives an overview of grounding approaches for WSMO.

Like OWL-S, WSMO allows to include preconditions and effects in service descriptions. However, WSMO provides more detailed concepts here. The semantics of a WSMO service are described as its *capability* which consists of four parts: preconditions, assumptions, effects, and postconditions. We now explain those parts using the example of a ticket booking service. This example is taken from [Feier & Domingue 2005].

WSMO preconditions specify the information that must be available in order to execute the service, e.g. the initial balance on the credit card used to book the ticket. WSMO assumptions specify the literals that must hold in the world state for the service to be executed. An example is a literal which specifies that the credit card is valid. Our PDDL preconditions (which are consistent with the naming convention in artificial intelligence) can thus be mapped to either WSMO preconditions or WSMO assumptions. WSMO effects, on the other hand, specify the world state after the execution of the service. In the ticket booking example, an effect is that the balance on the credit card is lowered by the price of the ticket.

WSMO postconditions describe the information that is present after the execution of the service. An example is the reservation for the trip as a result of booking the ticket. “Our” effects can therefore be modeled as WSMO effects or WSMO postconditions.

In contrast to OWL-S, WSMO explicitly encourages to define predicates and variable types that occur in preconditions, assumptions, effects and postconditions in ontologies. Names of predicates and variable types are URIs, i.e. links to these ontologies. Therefore, WSMO not only allows to describe the syntax, but also the semantics of operators. To align different ontologies, WSMO provides the concept of mediators. They ensure that terminology mismatches between ontologies can be resolved. Furthermore, WSMO has an extra concept for goals. This allows to formulate the observed intentions of the users directly as goals. Of particular importance for us is that WSMO is expressive enough to support the constructs we need for the AdDCo algorithm: function-free first-order literals and universal quantification. Therefore, WSMO seems to be better suited for describing operators than OWL-S. Our *ShowDoc* operator is shown as a WSMO service description in Figure 7.2.

WSMO service descriptions can be edited using WSMO Studio [Dimitrov *et al.* 2007]. It consists of a set of Eclipse plugins and is still in prototype stage. There is no tutorial available yet, and the documentation does not provide much help to a first-time user.

7.2.3 DSD (DIANE Service Description)

Like OWL-S and WSMO, DSD [Klein 2004] allows to describe the semantics of web services via preconditions and effects. All concepts in these service descriptions must be defined in DSD ontologies. Unlike effects, preconditions in DSD are not conjunctions of literals, but lists of entities that must be present for the service to be executed. To execute a service *BuyBook*, a customer must specify a book to buy, an account at the book selling web site and a credit card. So DSD preconditions rather specify information the service requires to be executed than literals that must hold in the world state (very much like WSMO preconditions). Thus, DSD preconditions are less expressive than the preconditions we need for our action descriptions. Therefore, DSD is better suited for domains that require matchmaking [Küster *et al.* 2007] than for ad-hoc ensembles that require run-time

```
namespace {_"http://example.org/ShowDoc#",
  dc      _"http://purl.org/dc/elements/1.1#",
  pr      _"http://example.org/praedikate#",
  dev     _"http://example.org/devices#"}

webService _"http://example.org/ShowDocWebService"

importsOntology _"http://example.org/ShowDocOntology"

capability ShowDocCapability

  sharedVariables {?Doc, ?Canv, ?OtherDoc}

  assumption
     nonFunctionalProperties
        dc#description hasValue "For a projector to be able to project a
        document, there must be a canvas that can be projected on.
        This canvas must be lowered. In addition, there must be a
        document to be projected."
    endNonFunctionalProperties
    definedBy
       ?Doc memberOf dev#Document
       and
       ?Canv memberOf dev#Canvas
       and
       ?OtherDoc memberOf dev#Document
       and
       ?Doc != ?OtherDoc
       and
       pr#SentToDisp(?Doc, Projector1)
       and
       pr#CanvasDown(?Canv).

  effect
    nonFunctionalProperties
       dc#description hasValue "If the action is executed, the following
       conditions hold: The document is shown on the canvas. If the
       projector has been projecting a different document onto the
       canvas, this document is not visible anymore."
    endNonFunctionalProperties
    definedBy
       pr#DocShown(?Doc, ?Canv)
       and
       forall ?OtherDoc (not pr#DocShown(?OtherDoc, ?Canv)).
```

Figure 7.2: Service description in WSMO.

strategy synthesis. In DSD the service grounding does not rely on any ”extern“ languages such as WSDL, but is included in the DSD standard. Thus, DSD allows to specify ontologies, service descriptions and service grounding all in one coherent formalism. This is a huge advantage over other languages such as OWL-S and WSMO as it helps to make services of different origin interoperable. DSD is still in an experimental state: It is hard to find complete examples of service descriptions in DSD and there is no single easy-to-understand representation. Several representations for DSD descriptions exist in parallel, including a graphical notation called g-dsd, a textual notation called f-dsd, and a Java representation called j-dsd. Some of these representations can be translated into each other, some cannot. Furthermore, there is little tool support at the moment. G-dsd, for example, can be edited via a set of Microsoft Visio templates which are not publicly available.

7.2.4 WSDL-S (Web Service Semantics) and SAWSDL (Semantic Annotations for WSDL)

WSDL-S [Akkiraju *et al.* 2005] allows for semantic annotations, but does not specify a format for these annotations. It allows the user to annotate WSDL services with ontologies in any language. This makes it hard, if not impossible, to foster interoperability of services of different origin. Preconditions and effects can be described in external languages such as SWRL [Horrocks *et al.* 2004]. This causes the same problems as OWL-S preconditions and effects. Another drawback is that WSDL-S allows just one precondition and one effect per service at maximum. To describe conjunctions of preconditions and effects, one has to specify a high-level precondition and resolve it using an ontology. SAWSDL [Kopecký *et al.* 2007] is the successor of WSDL-S. Unlike WSDL-S, SAWSDL does not provide a means to specify preconditions and effects which makes it unsuitable for our purposes. Both SAWSDL and WSDL-S service descriptions can be edited using the Radiant annotation tool [Gomadam *et al.* 2005] which provides an easy-to-use interface for augmenting existing WSDL files with semantic annotations.

As WSDL-S and SAWSDL are extensions to standard WSDL, services described in WSDL can be used straightforward. This is an advantage over OWL-S and WSMO where the grounding must be realized by a special mapping.

7.2.5 Benefits and shortcomings

In summary, one can say that each semantic web language has its benefits and shortcomings, but neither of them seems perfectly suitable for describing device actions. The advantages of OWL-S and WSMO services are that they can accomodate preconditions and effects. For us it is especially important that OWL-S and WSMO services allow for the expressivity required for the AdDCo algorithm. However, there are a number of drawbacks: In OWL-S, preconditions and effects are just represented as string literals. OWL-S as well as WSMO do not specify a grounding. The other semantic web languages have even more shortcomings: Preconditions in DSD are not literals that must be fulfilled in the world, but entities that must be present for the service to be executed. WSDL-S and SAWSDL do not specify a format for semantic annotations, and SAWSDL does not support preconditions and effects. All lack sufficient tool support. Of all these languages, WSMO seems the most promising as it provides a useful means of representing "our" preconditions (which correspond to WSMO preconditions or assumptions) and effects (which can be modeled as WSMO effects or postconditions). Furthermore, WSMO allows to specify all concepts, including predicates and variable types, using ontologies. If a specification of service grounding was included in the standard, it might be worth considering for our domain.

7.3 Embracing the Semantic Web

Having elaborated on the question *in which form* action descriptions can be provided, the next question is *who* could provide them. Descriptions of device actions in the form of semantic web services might be written by device vendors and supplied with devices, just like it is common today for syntactic service descriptions, e.g. in UPnP [Jeronimo & Weast 2003]. This would enable the device cooperation we are aiming for: Any device supplied with such descriptions would be ready for ad-hoc cooperation.

One difficulty is that device vendors cannot foresee every possible context or use case. An example: A projector has the primary effect of showing a document. However, it has secondary effects that might not be apparent in the first place: It lights and heats a room and consumes power. Whether these effects are significant depends on the context of use. If the user wants to take into account power consumption, s/he has to modify the action descriptions accordingly. Thus, it would be useful to have several descriptions of a device action that could be used according to the context.

Device vendors cannot provide all of these action descriptions, but the user community could. Users can collaboratively create and refine service descriptions [Braun *et al.* 2007] and share them on the web. They could also refine the descriptions provided by the device vendors. The success of web 2.0 applications like Flickr and Wikipedia shows that users collaborate on the internet to generate content. Another area where collaboration yields impressing results is open source software.

Semantic web services have not yet gained acceptance by a broad user community. This is partly due to the fact that the languages for semantic web services are hard to read and use. Furthermore, convenient tools for editing them are still missing. Web 2.0 applications are successful among a large community because they have easy-to-use and intuitive user interfaces. Semantic web services require similar interfaces to become widespread. Of course, the community that is interested in Web 2.0 applications is undoubtedly much larger than the group of people that would be interested in writing action descriptions for devices. However, we believe that a new community of technophilic people that are both users *and* developers could arise here – similar to the open source community.

Another problem is the lack of large repositories of services that can be collaboratively edited. However, there are advances in that direction [Ankolekar *et al.* 2007]. One such project is the OPOSSum web site [Küster *et al.* 2008], a repository that allows the community to upload and collaboratively refine semantic web services. It is still in a very early stage of development, but already contains several test collections of service descriptions in OWL-S and WSDL. Unfortunately, no WSMO service descriptions are available yet.

7.4 Chapter Summary

In the previous chapters, we have introduced the AdDCo algorithm, which is a mechanism for run-time strategy synthesis. This kind of mechanism requires declarative descriptions of device actions. “Traditional” planning languages like

PDDL are not suited for this kind of description because they do not allow to define concepts using ontologies. Thus, interoperability of descriptions of different origin is not ensured. In this chapter, we have shown that semantic web services are in principle a favorable formalism for such descriptions. We have furthermore investigated which languages for semantic web services are currently available, and how well each of them is suited for describing device actions. We have found that several semantic web languages exist, namely OWL-S, WSMO, DSD, WSDL-S, and SAWSDL. Yet adoption by a large community is hampered by significant shortcomings of all of them, including weak or no means to specify preconditions and effects, complex syntax, and missing tool support. However, WSMO seems the most suitable of all semantic web languages with respect to smart environments. If there were tools to easily edit service descriptions and a place on the internet where device vendors and users could collaborate to produce them, the semantic web could help to bring the vision of intelligent environments to life.

Chapter 8

Conclusion and Outlook

Contents

Users in environments with a high number of technical devices are often overwhelmed – they wonder how to get those devices to do what they want. Therefore, it is necessary to assist users in such environments. This is especially important in *ad-hoc environments*, which consist of fixed devices as well as mobile devices. One approach to building systems that assist users in such environments is to divide them into an *intention analysis* part and a *strategy synthesis* part. The intention analysis infers the goals of the user, and the strategy synthesis searches for an action sequence the devices must carry out to fulfill those goals.

With this thesis, we have tried to investigate how successful strategy synthesis in smart ad-hoc environments can be achieved. At first, we wanted to find out which usage situations typically occur in such environments. We have therefore performed a domain analysis with the help of typical scenarios from the literature. This enabled us to extract a set of requirements for strategy synthesis. We have furthermore formulated two main questions to guide our work:

1. Is it possible to engineer a system for ad-hoc device cooperation in smart environments in a fully distributed fashion?
2. Do users accept the assistance such a system can provide?

An analysis of existing approaches for strategy synthesis has then shown that none of them is suited for ad-hoc environments because none fulfills all requirements. Specifically, there is currently no distributed control strategy for smart environments. To show that distributed strategy synthesis in smart ad-hoc environments is in principle possible, we have developed the AdDCo algorithm. It fulfills all our requirements apart from *rationality*: Because the algorithm is based on local knowledge only, it sometimes generates suboptimal action sequences. Hence, it only partially fulfills the requirement *rationality*. We have then carried out a quantitative user study to answer two important questions:

- Do users accept an assistance system that sometimes shows suboptimal behavior?
- Which contextual factors affect user acceptance?

The latter is especially important because users in Ubiquitous Computing are situated in "real life" rather than sitting in front of a desktop computer. Therefore, contextual factors can influence user acceptance. Furthermore, we have tried to answer the question how and by whom the declarative action descriptions needed for a distributed strategy synthesis mechanism can be provided.

8.1 Key Results

The main results of this thesis are:

1. In principle, it is possible to engineer a fully decentralized system for ad-hoc device cooperation in smart environments.
2. Users accept such a system even if it provides suboptimal assistance. However, acceptance is influenced by several contextual factors.

In the following, we provide a more detailed account of these results. From our scenario analysis, we have extracted the following set of requirements an appropriate strategy synthesis mechanism should fulfill:

- spontaneity
- action sequences
- rationality
- flexibility
- robustness
- support for persistent actions
- distributedness (run-time modularization and design-time modularization)

With the help of the aforementioned scenarios, we have furthermore formed the hypothesis that in terms of Wandke's assistance framework *informative exe-*

cution assistance is the best level of assistance in smart environments, that *fixed assistance* is appropriate and that users prefer *active assistance*. However, our quantitative user study has then shown that this hypothesis is only partially correct: *Informative execution assistance* is not always the best level of assistance. In certain situations, users prefer another level that gives them more control. Our conclusion is that future assistance systems should be able to adapt their level of assistance. This also implies that, concerning adjustment, *fixed assistance* is not preferable over *adaptable* or *adaptive assistance*. Because the user may not always have the time and cognitive resources to adapt the system her-/himself, we argue that *adaptive assistance* is most feasible. Concerning initiative, our hypothesis was correct: *Active assistance* is preferable over *passive assistance*.

We have argued that neither purely *deliberative* nor purely *reactive* control strategies are suited for smart ad-hoc environments. A strategy synthesis mechanism that fulfills the requirements discussed above must be *hybrid*. Currently, two kinds of hybrid architectures exist: *horizontally* and *vertically layered* architectures. None of them is fully suited for the special usage situation in smart environments. We have therefore introduced a third kind, the *temporally layered* architecture: A deliberative step that prunes irrelevant actions is followed by a reactive step.

We have described the AdDCo algorithm, which is an implementation of the temporally layered architecture. It is the first distributed strategy synthesis mechanism for smart environments and is based on Maes' action selection mechanism, which is a reactive mechanism. Like Maes' approach, the AdDCo algorithm supports *goal-based interaction*, which is a favorable interaction paradigm for smart environments. However, in contrast to Maes' algorithm, our version can be distributed over the devices in a smart environment. Due to the deliberative step we included, it is also more goal-oriented.

In this form, the algorithm fulfills the requirements *spontaneity*, *action sequences*, and *robustness*. To enhance *flexibility*, we have then added support for universally quantified effects. We have furthermore added support for *persistent actions* and have described how to hide information in order to enhance *design-time modularization*, which is a part of the requirement *distributedness*. The AdDCo algorithm now fulfills all requirements, apart from *rationality*, which is only partially fulfilled. Due to the limited knowledge of the algorithm, this requirement cannot be fulfilled completely: The algorithm can produce suboptimal action sequences.

In our quantitative user study we were able to show that:

- users in smart environments accept even suboptimal assistance,
- users experiencing a certain task load find imperfect assistance systems more useful than users with a low task load,
- users with some experience in using such a system find it easier to use than novices,
- if automatic assistance exhibits suboptimal behavior, users find it less useful than when it exhibits optimal behavior. However, if it offers enough benefit compared to manual control of an environment, people prefer automatic assistance even if it is suboptimal.

Our investigations regarding the question how declarative action descriptions should be provided have yielded the following results: Semantic web service languages are, in principle, suitable for such action descriptions. They could be provided by the device manufacturers and refined and extended by the user community. However, in practice, all existing languages suffer from weak or no means to represent preconditions and effects, complex syntax, and missing tool support. However, of all currently available semantic web languages, WSMO seems the most promising with respect to smart environments.

8.2 Outlook

This thesis has addressed the question how strategy synthesis in ad-hoc environments can be achieved in principle. We have focused on finding out which requirements a successful strategy synthesis mechanism should fulfill and on demonstrating that it is possible to engineer a system which meets those requirements and which is accepted by the users. Hence, this thesis has a very explorative character. There are several questions we could only marginally touch, but which are worth to be studied more comprehensively in order to gain a deeper understanding of our proposed approach.

The AdDCo algorithm has the property that action sequences emerge from the interactions of its components. The drawback here is that the behavior of the system can be hard to predict and understand. Therefore, it would be beneficial to develop a mathematical model of the AdDCo algorithm. As Maes has stated, her algorithm can be modeled as a system of differential equations. However, Maes also remarked that such a system is too complex to solve analytically. The same probably holds for the AdDCo algorithm. However, differential equations are only one possibility to model the algorithm. Yet when deciding how to model the AdDCo algorithm, one has to keep in mind that it is fully distributed and nodes in the network may join and leave anytime. Therefore, any modelling paradigm that cannot incorporate such dynamicity, such as for instance Petri nets, is not suited for the AdDCo algorithm. Finding an appropriate modelling mechanism is thus not trivial and can be considered as an open research question. However, with the help of a mathematical model, we could answer fundamental questions such as:

- If there is a possible action sequence, does the AdDCo algorithm always find it?
- If the algorithm finds an action sequence, how long is this sequence in comparison to the shortest possible sequence?
- How should the world be modeled (i.e., how should actions be described) such that the algorithm finds a (short) action sequence?

Another issue is the complexity of the AdDCo algorithm. In the worst case, the algorithm does not have polynomial running time. This follows from the fact that it does not even terminate if no solution exists (i.e. if no action sequence can be constructed that fulfills the goals). Hence, our algorithm does not guarantee to find a solution, neither does it guarantee polynomial running time. However, consider the fact that the AdDCo algorithm solves planning problems. Bylander has shown that planning is PSPACE-complete [Bylander 1991]. Hence, under the assumption that $P \subset PSPACE$ (which most researchers believe to be the case), no polynomial time algorithm can be constructed that always finds a solution if one exists. Nevertheless, our observations have shown that the AdDCo algorithm performs well on the majority of scenarios. Hence, the worst-case view seems too pessimistic and an average case analysis would be desirable. Yet this seems to be much harder. For such an analysis, one would need a realistic model of random inputs.

Another interesting question is how much shorter the action sequences generated by the reduced network described in Section 4.4.8 are in comparison to those generated by the full network. Our focus was on improving the algorithm, which can clearly be achieved by reducing the network. We have seen this in the evaluation in Section 4.5: The AdDCo algorithm performed better than the PM algorithm, and a big part of this improvement is due to the reduction of the network. However, it would be interesting to study in more depth how many operators the reduction can exclude from the network on average.

Furthermore, we have introduced user assistance as a two-stage process consisting of an *intention analysis* component and a *strategy synthesis* component, but have not dealt with the intention analysis at all. Throughout the thesis, we have assumed that the intention analysis correctly provides the goals the user actually wants to be fulfilled. This is key for the success of the strategy synthesis. In practice, however, this assumption does not yet hold: The intention analysis is a separate field which is currently under active research. Much work is left to be done until the intention analysis can actually provide user goals in the granularity required by the strategy synthesis.

Obviously, with this thesis we could merely address a few of the challenges on the way to Mark Weiser's vision of calm, unobtrusive computing. We have been able to show that spontaneous, decentralized device cooperation is indeed feasible. We have found out that users do not expect perfect assistance, but are content with a system that makes their lives a little easier. These are motivating results that future researchers in the field of smart ad-hoc environments can build upon.

CHAPTER 9

Appendix

9.1 Notational Conventions

Let *String* denote the set of character strings. Typewriter font – `abc` – denotes elements of *String*, literal text in the object language. Often, we will want to use meta language variables in object language text – for instance `(and` a b`)`, where a, b : *String* are meta language variables representing object language text strings. To explicitly delimit an object language text string containing meta language variables, we use Quine quasi-quotes [van Orman Quine 2003] and write this as $\ulcorner$`(and` a b`)`$\urcorner$. Thus, if a = `xy` and b = `pq` then $\ulcorner$`(and` a b`)`$\urcorner$ is the text string `(and xy pq)`. The Quine quasi quotes always result in text object language text strings.

The object language string `(not ?a)` $\in$ *String* is a sequence of eight characters, starting with the character `(` and ending with the character `)` (the fifth character is the space character). However, usually, we will want to discuss the object language at the level of the abstract syntax tree, which we subsume in the set *Expression*. Here, we will use the notation $[\![$`(not ?a)`$]\!]$ to denote the abstract syntax tree for the object language string `(not ?a)`. One can think of $[\![\cdot]\!]$: *String* $\rightarrow$ *Expression* as a parser function that gives the abstract syntax tree for the argument string. In abstract syntax trees we too will allow meta-language variables and expressions such as $[\![$`(not` a`)`$]\!]$. For a start, the meta language expression a denotes an object language character string, such that $[\![$`(not` a`)`$]\!]$ is the abstract syntax tree we get by parsing the object language string $\ulcorner$`(not` a`)`$\urcorner$.

Let *print* : *Expression* $\rightarrow$ *String* be a function that gives a character string representation for an abstract syntax tree such that for an abstract syntax tree t we have the identity $[\![(print\ t)]\!] = t$. Then $[\![$`(not` $(print\ t)$`)`$]\!]$ is the abstract syntax tree we get by parsing the string $\ulcorner$`(not` a`)`$\urcorner$, where a : *String* and $a = (print\ t)$. As a slight inconsistency, we will simply write $[\![$`(not` t`)`$]\!]$, where $[\![$`(not` $(print\ t)$`)`$]\!]$

would have been correct. (Conceptually, $[\![$`(not` t`)`$]\!]$ describes an operation at the level of abstract syntax trees rather than at the level of source language strings: $[\![$`(not` t`)`$]\!]$ is the tree we get by taking the abstract syntax tree of $[\![$`(not` $\cdot$`)`$]\!]$ and replacing the tree for $\cdot$ by the tree t.)

Furthermore, we will often omit $[\![$ and $]\!]$. For example, we will write `(not ?a)` instead of $[\![$`(not ?a)`$]\!]$. There are two reasons for this: First, it impropves readability. Second, as object language is typeset in typewriter font, it can easily be distinguished from meta language.

9.2 Example: Instantiating an Operator Scheme Containing a Universally Quantified Effect

We have a domain τ with two types: *Document* and *Notebook*. *Document* has seven objects, *Notebook* has one object:

$$\begin{aligned}\tau = \{&\texttt{Document} \mapsto \{\texttt{Doc1,Doc2,Doc3,Doc4,Doc5,Doc6,Doc7}\},\\ &\texttt{Notebook} \mapsto \{\texttt{NB1}\}\}\end{aligned} \tag{9.1}$$

Furthermore, we have the following operator scheme:

$$\sigma = \textit{Maximize}$$

```
  = (:parameters (?x - Document ?n - Notebook)
     :precondition (Hosts ?x ?n)
     :effect (and (forall (?x - Document)(not (isMax ?x ?n)))
                  (isMax ?x ?n)))
```
(9.2)

Note that for the binding $\beta_1 = \{\texttt{?x} \mapsto \texttt{Doc1}, \texttt{?n} \mapsto \texttt{NB1}\}$, which we will use as an example in the following, this operator scheme semantically corresponds to the *Maximize* operator described in (5.1). Syntactically, it is different because in our notation we assume that effects are executed sequentially. Hence, we make use of this fact. However, this behavior is not supported by standard PDDL, thus the *Maximize* operator in (5.1) expresses the same semantics using a *when* construct. Furthermore, note that the effect contains a first order term whose declaration `(?x - Document)` introduces a local variable `?x` that shadows the parameter variable `?x`.

For instantiating the scheme σ according to (4.37), we first compute

$$\begin{aligned} \mathit{bindings}\ (\mathit{decl}(\sigma))\ \tau = \mathit{bindings}\ &\{\texttt{?x} \mapsto \texttt{Document}, \texttt{?n} \mapsto \texttt{Notebook}\} \\ &\{\texttt{Document} \mapsto \{\texttt{Doc1,Doc2,Doc3,Doc4,} \\ &\qquad \texttt{Doc5,Doc6,Doc7}\}, \\ &\ \texttt{Notebook} \mapsto \{\texttt{NB1}\}\} \end{aligned} \tag{9.3}$$

Employing (4.30):

$$\begin{aligned} =\{&\{\texttt{?x} \mapsto \texttt{Doc1}, \texttt{?n} \mapsto \texttt{NB1}\}, \\ &\{\texttt{?x} \mapsto \texttt{Doc2}, \texttt{?n} \mapsto \texttt{NB1}\}, \\ &\{\texttt{?x} \mapsto \texttt{Doc3}, \texttt{?n} \mapsto \texttt{NB1}\}, \\ &\{\texttt{?x} \mapsto \texttt{Doc4}, \texttt{?n} \mapsto \texttt{NB1}\}, \\ &\{\texttt{?x} \mapsto \texttt{Doc5}, \texttt{?n} \mapsto \texttt{NB1}\}, \\ &\{\texttt{?x} \mapsto \texttt{Doc6}, \texttt{?n} \mapsto \texttt{NB1}\}, \\ &\{\texttt{?x} \mapsto \texttt{Doc7}, \texttt{?n} \mapsto \texttt{NB1}\}\} \end{aligned} \tag{9.4}$$

For each $\beta \in \mathit{bindings}\ (\mathit{decl}(\sigma))\ \tau$ we then compute the corresponding instantiated operator σ_β as `(:precondition` $\mathit{subst}_\beta(\mathit{pre}(\sigma))$ `:effect` $\mathit{subst}_\beta(\mathit{eff}(\sigma))$`)`. We will do this explicitly for the first binding $\beta_1 = \{\texttt{?x} \mapsto \texttt{Doc1}, \texttt{?n} \mapsto \texttt{NB1}\}$:

$$\begin{aligned} \sigma_{\beta_1} =&\texttt{(:precondition }\mathit{subst}_{\beta_1}(\mathit{pre}(\sigma)) \\ &\texttt{:effect }\mathit{subst}_{\beta_1}(\mathit{eff}(\sigma))\texttt{)} \end{aligned} \tag{9.5}$$

Inserting terms:

$$\begin{aligned} =&\texttt{(:precondition }\mathit{subst}_{\beta_1}\texttt{(Hosts ?x ?n)} \\ &\texttt{:effect }\mathit{subst}_{\beta_1}\texttt{(and} \\ &\qquad\qquad \texttt{(forall (?x - Document)(not (isMax ?x ?n)))} \\ &\qquad\qquad \texttt{(isMax ?x ?n)))} \end{aligned} \tag{9.6}$$

By (4.14):

$$=\texttt{(:precondition (Hosts } (subst_{\beta_1}\texttt{?x})\ (subst_{\beta_1}\texttt{?n})\texttt{)}$$

```
:effect subst_β1(and
              (forall (?x - Document)(not (isMax ?x ?n)))
              (isMax ?x ?n)))                              (9.7)
```

By (4.12) and the fact that $\beta_1 = \{\texttt{?x} \mapsto \texttt{Doc1}, \texttt{?n} \mapsto \texttt{NB1}\}$:

```
=(:precondition (Hosts Doc1 NB1)
  :effect subst_β1(and
              (forall (?x - Document)(not (isMax ?x ?n)))
              (isMax ?x ?n)))                              (9.8)
```

By (4.16):

```
=(:precondition (Hosts Doc1 NB1)
  :effect (and
           (subst_β1(forall (?x - Document)(not (isMax ?x ?n))))
           (subst_β1(isMax ?x ?n))))                       (9.9)
```

Substituting in `isMax` using (4.14), (4.12) and the fact that $\beta_1 = \{\texttt{?x} \mapsto \texttt{Doc1}, \texttt{?n} \mapsto \texttt{NB1}\}$ (just as we did for `Hosts`):

```
=(:precondition (Hosts Doc1 NB1)
  :effect (and
           (subst_β1(forall (?x - Document)(not (isMax ?x ?n))))
           (isMax Doc1 NB1)))                              (9.10)
```

Using (5.5):

```
=(:precondition (Hosts Doc1 NB1)
    :effect (and
            (forall (?x - Document)($subst_{\beta'_1}$(not (isMax ?x ?n))))
            (isMax Doc1 NB1)))                                        (9.11)
```

By (4.15) and (4.14):

```
  =(:precondition (Hosts Doc1 NB1)
    :effect (and
            (forall (?x - Document)
                    (not (isMax ($subst_{\beta'_1}$ ?x) ($subst_{\beta'_1}$ ?n))))
            (isMax Doc1 NB1)))                                        (9.12)
```

Using $\beta'_1 = (\text{dom}\{\texttt{?x} \mapsto \texttt{Doc1}\}) \lhd \beta_1 = \{\texttt{?x}\} \lhd \{\texttt{?x} \mapsto \texttt{Doc1}, \texttt{?n} \mapsto \texttt{NB1}\} = \{\texttt{?n} \mapsto \texttt{NB1}\}$

```
  =(:precondition (Hosts Doc1 NB1)
    :effect (and
            (forall (?x - Document)
                    (not (isMax ($subst_{\{?n \mapsto NB1\}}$?x) ($subst_{\{?n \mapsto NB1\}}$?n))))
            (isMax Doc1 NB1)))                                        (9.13)
```

By (4.12), second case:

```
  =(:precondition (Hosts Doc1 NB1)
    :effect (and
            (forall (?x - Document)
                    (not (isMax ?x ($subst_{\{?n \mapsto NB1\}}$?n))))
            (isMax Doc1 NB1)))                                        (9.14)
```

By (4.12), first case:

```
  =(:precondition (Hosts Doc1 NB1)
    :effect (and
            (forall (?x - Document)
                    (not (isMax ?x NB1))))
            (isMax Doc1 NB1)))                                    (9.15)
```

Note that an instantiated operator may still include variables in the first order effect terms. Let us now compute the effective effect of σ_{β_1} via (5.13):

$$\begin{aligned} ceff_\tau(\sigma_{\beta_1}) &= cff(dq_\tau(eff(\sigma_{\beta_1}))) && (9.16)\\ &= cff(dq_\tau(\texttt{(and} \\ &\qquad \texttt{(forall (?x - Document)} \\ &\qquad\qquad \texttt{(not (isMax ?x NB1)))} \\ &\qquad \texttt{(isMax Doc1 NB1))))} && (9.17)\end{aligned}$$

By (5.9):

$$\begin{aligned} &= cff(\texttt{and } (dq_\tau(\texttt{forall (?x - Document)} \\ &\qquad\qquad\qquad \texttt{(not (isMax ?x NB1))))} \\ &\qquad (dq_\tau(\texttt{isMax Doc1 NB1)))} && (9.18)\end{aligned}$$

By (5.10):

$$\begin{aligned} &= cff(\texttt{and } (dq_\tau(\texttt{forall (?x - Document)} \\ &\qquad\qquad\qquad \texttt{(not (isMax ?x NB1))))} \\ &\qquad \texttt{(isMax Doc1 NB1))} && (9.19)\end{aligned}$$

By (5.11):

$$\begin{aligned} &= cff(\texttt{and } (expand_\tau(\texttt{forall (?x - Document)} \\ &\qquad\qquad\qquad \texttt{(not (isMax ?x NB1))))} \\ &\qquad \texttt{(isMax Doc1 NB1))} && (9.20)\end{aligned}$$

By (5.7), we need to compute the bindings for the declaration {`?x` ↦ `Document`} in τ, which gives β_1 = {`?x` ↦ `Doc1`}, β_2 = {`?x` ↦ `Doc2`}, ..., β_7 = {`?x` ↦ `Doc7`}:

= *cff* `(and`
  ($subst_{\beta_1}$`(not (isMax ?x NB1)))`
  ($subst_{\beta_2}$`(not (isMax ?x NB1)))`
  ($subst_{\beta_3}$`(not (isMax ?x NB1)))`
  ($subst_{\beta_4}$`(not (isMax ?x NB1)))`
  ($subst_{\beta_5}$`(not (isMax ?x NB1)))`
  ($subst_{\beta_6}$`(not (isMax ?x NB1)))`
  ($subst_{\beta_7}$`(not (isMax ?x NB1)))`
  `(isMax Doc1 NB1))` (9.21)

Performing the substitutions:

```
= cff(and
    (not (isMax Doc1 NB1))
    (not (isMax Doc2 NB1))
    (not (isMax Doc3 NB1))
    (not (isMax Doc4 NB1))
    (not (isMax Doc5 NB1))
    (not (isMax Doc6 NB1))
    (not (isMax Doc7 NB1))
    (isMax Doc1 NB1))                    (9.22)
```

Now, the last step is to reduce this to a conflict-free formula by *cff*. By (4.20):

```
= cff((and
     (not (isMax Doc1 NB1))
     (not (isMax Doc2 NB1))
     (not (isMax Doc3 NB1))
```

$$\begin{aligned}
&\texttt{(not (isMax Doc4 NB1))}\\
&\texttt{(not (isMax Doc5 NB1))}\\
&\texttt{(not (isMax Doc6 NB1))}\\
&\texttt{(not (isMax Doc7 NB1))}\\
&\upharpoonleft\texttt{(not (isMax Doc1 NB1))))}\\
&\frown\texttt{(isMax Doc1 NB1)}
\end{aligned} \tag{9.23}$$

After successively applying (4.20) and (4.19) and by effect of $\upharpoonleft$ we arrive at:

$$\begin{aligned}
= \texttt{(and }&\texttt{(not (isMax Doc2 NB1))}\\
&\texttt{(not (isMax Doc3 NB1))}\\
&\texttt{(not (isMax Doc4 NB1))}\\
&\texttt{(not (isMax Doc5 NB1))}\\
&\texttt{(not (isMax Doc6 NB1))}\\
&\texttt{(not (isMax Doc7 NB1)))}\\
&\frown\texttt{(isMax Doc1 NB1)}
\end{aligned} \tag{9.24}$$

By effect of $\frown$:

$$\begin{aligned}
= \texttt{(and }&\texttt{(not (isMax Doc2 NB1))}\\
&\texttt{(not (isMax Doc3 NB1))}\\
&\texttt{(not (isMax Doc4 NB1))}\\
&\texttt{(not (isMax Doc5 NB1))}\\
&\texttt{(not (isMax Doc6 NB1))}\\
&\texttt{(not (isMax Doc7 NB1))}\\
&\texttt{(isMax Doc1 NB1))}
\end{aligned} \tag{9.25}$$

Hence, the complete instantiated operator is the following:

$$\begin{aligned}
\sigma_{\beta_1} = &\texttt{(:precondition (Hosts Doc1 NB1)}\\
&\quad\texttt{:effect (and (not (isMax Doc2 NB1))}
\end{aligned}$$

```
(not (isMax Doc3 NB1))
(not (isMax Doc4 NB1))
(not (isMax Doc5 NB1))
(not (isMax Doc6 NB1))
(not (isMax Doc7 NB1))
(isMax Doc1 NB1)))                    (9.26)
```

Bibliography

[Akkiraju *et al.* 2005] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth and K. Verma. *Web Service Semantics – WSDL-S*. http://www.w3.org/Submission/WSDL-S/, 2005. 186, 192

[Amigoni *et al.* 2005] Francesco Amigoni, Nicola Gatti, C. Pinciroli and Manuel Roveri. *What planner for ambient intelligence applications?* IEEE Transactions on Systems, Man, and Cybernetics, Part A, vol. 35, no. 1, pages 7–21, 2005. v, 15, 18, 40, 144, 183

[Ankolekar *et al.* 2007] Anupriya Ankolekar, Markus Krötzsch, Thanh Tran and Denny Vrandecic. *The two cultures: mashing up web 2.0 and the semantic web*. In WWW '07: Proceedings of the 16th international conference on World Wide Web, pages 825–834, New York, NY, USA, 2007. ACM. 194

[Arkin & Mackenzie 1994] Ronald C. Arkin and Douglas C. Mackenzie. *Planning to Behave: A Hybrid Deliberative/Reactive Robot Control Architecture for Mobile Manipulation*. In International Symposium on Robotics and Manufacturing, pages 5–12, 1994. 57, 112

[Bellotti & Edwards 2001] Victoria Bellotti and Keith Edwards. *Intelligibility and accountability: Human considerations in context-aware systems*. Hum.-Comput. Interact., vol. 16, no. 2, pages 193–212, 2001. 30

[Beni & Wang 1989] Gerardo Beni and Jing Wang. *Swarm Intelligence in Cellular Robotic Systems*. In NATO Advanced Workshop on Robots and Biological Systems, pages 703–712, Tuscany, Italy, 1989. 46

[Blum & Furst 1995] Avrim Blum and Merrick Furst. *Fast Planning Through Planning Graph Analysis*. In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95), pages 1636–1642, 1995. 41, 144

[Bortz & Döring 2006] Jürgen Bortz and Nicola Döring. *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. Springer, 2006. 149, 150, 159, 166, 173

[Braun *et al.* 2007] Simone Braun, Andreas Schmidt and Valentin Zacharias. *Ontology Maturing with Lightweight Collaborative Ontology Editing Tools*. In Proceedings of 4th Conference for Professional Knowledge - Experiences and Visions, Potsdam, Germany, 2007. 194

[Brooks 1990] Rodney A. Brooks. *A robust layered control system for a mobile robot*. Artificial intelligence at MIT: expanding frontiers, pages 2–27, 1990. 48

[Brumitt *et al.* 2000] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern and Steven A. Shafer. *EasyLiving: Technologies for Intelligent Environments*. In Peter J. Thomas and Hans-Werner Gellersen, editors, HUC, volume 1927 of *Lecture Notes in Computer Science*, pages 12–29. Springer, 2000. 48

[Butter *et al.* 2007] Thomas Butter, Markus Aleksy, Philipp Bostan and Martin Schader. *Context-aware User Interface Framework for Mobile Applications*. In Proceedings of the 27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07), page 39, Los Alamitos, CA, USA, 2007. IEEE Computer Society. 178

[Bylander 1991] Tom Bylander. *Complexity Results for Planning*. In IJCAI, pages 274–279, 1991. 201

[Castro *et al.* 2009] Mariana N. Castro, Daniel E. Vigo, Elvina M. Chu, Rodolfo D. Fahrer, Delfina D. de Achával, Elsa Y. Costanzo, Ramón C. Leiguarda, Martín Nogués, Daniel P. Cardinali and Salvador M. Guinjoan. *Heart rate variability response to mental arithmetic stress is abnormal in first-degree relatives of individuals with schizophrenia*. Schizophrenia Research, vol. 109, no. 1, pages 134–140, 2009. 157

[Chen & Yang 2005] Liming Chen and Xueqiang Yang. *Applying AI Planning to Semantic Web Services for Workflow Generation*. In SKG '05: Proceedings of the First International Conference on Semantics, Knowledge and Grid, page 65, Washington, DC, USA, 2005. IEEE Computer Society. 184

[Coen 1997] Michael H. Coen. *Building Brains for Rooms: Designing Distributed Software Agents*. In AAAI/IAAI, pages 971–977, 1997. v, 15, 17, 49

[Connelly & Khalil 2004] Kay Connelly and Ashraf Khalil. *On Negotiating Automatic Device Configuration in Smart Environments*. Pervasive Computing and Communications Workshops, IEEE International Conference on, vol. 0, page 213, 2004. v, 15, 21

[Connelly 2007] Kay Connelly. *On Developing a Technology Acceptance Model for Pervasive Computing*. In Ubiquitous System Evaluation (USE) – a workshop at the Ninth International Conference on Ubiquitous Computing, September 2007. 148, 180

[Corkill 1979] Daniel D. Corkill. *Hierarchical planning in a distributed environment*. In IJCAI'79: Proceedings of the 6th international joint conference on Artificial intelligence, pages 168–175, San Francisco, CA, USA, 1979. Morgan Kaufmann Publishers Inc. 42

[Cummings *et al.* 2007] M.L. Cummings, S. Bruni, S. Mercier and P.J. Mitchell. *Automation Architecture for Single Operator, Multiple UAV Command and Control*. The International C2 Journal, vol. 1, no. 2, pages 1–24, 2007. 6

[Das *et al.* 2002] Sajal K. Das, Diane J. Cook, Amiya Bhattacharya, Edwin O. Heierman III and Tze-Yun Lin. *The Role of Prediction Algorithms in the MavHome Smart Home Architecture*. IEEE Wireless Communications, vol. 9, no. 6, pages 77–84, December 2002. v, 15, 19, 49

[Davis *et al.* 1989] Fred D. Davis, Richard P. Bagozzi and Paul R. Warshaw. *User acceptance of computer technology: a comparison of two theoretical models*. Manage. Sci., vol. 35, no. 8, pages 982–1003, 1989. 142

[Davis 1989] Fred D. Davis. *Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology*. MIS Quarterly, vol. 13, no. 3, pages 319–339, 1989. 148, 176

[Decugis & Ferber 1998] Vincent Decugis and Jacques Ferber. *An extension of Maes' Action Selection Mechanism for Animats*. In SAB'98, pages 153–158, Cambridge, MA, USA, 1998. MIT Press. 53

[Dimitrov *et al.* 2007] Marin Dimitrov, Alex Simov, Vassil Momtchev and Mihail Konstantinov. *WSMO Studio - A Semantic Web Services Modelling Environment for WSMO*. In Enrico Franconi, Michael Kifer and Wolfgang May, editors, ESWC, volume 4519 of *Lecture Notes in Computer Science*, pages 749–758. Springer, 2007. 186, 190

[Dorer 1999] Klaus Dorer. *Behavior Networks for Continuous Domains using Situation-Dependent Motivations*. In IJCAI, pages 1233–1238, 1999. 52

[Dorigo *et al.* 1996] Marco Dorigo, Vittorio Maniezzo and Alberto Colorni. *The Ant System: Optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics-Part B, vol. 26, pages 29–41, 1996. 46

[Durfee & Lesser 1991] E.H. Durfee and V.R. Lesser. *Partial global planning: a coordination framework for distributed hypothesis formation*. Systems, Man and Cybernetics, IEEE Transactions on, vol. 21, no. 5, pages 1167–1183, Sep/Oct 1991. 42

[Elenius *et al.* 2005] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati and Rukman Senanayake. *The OWL-S Editor - A Development Tool for Semantic Web Services*. In Proceedings of the Second European Semantic Web Conference, pages 78–92, Heraklion, Crete, Greece, 2005. 186, 189

[Erl 2007] Thomas Erl. *SOA Principles of Service Design*. Prentice Hall, Upper Saddle River, NJ, USA, 2007. 29

[Feier & Domingue 2005] Cristina Feier and John Domingue. *D3.1v0.1 WSMO Primer*. http://www.wsmo.org/TR/d3/d3.1/v0.1, April 2005. Accessed April 26, 2010. 189

[Fensel *et al.* 2006] Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman and John Domingue. *Enabling Semantic Web Services – The Web Service Modeling Ontology*. Springer, 2006. 188

[Ferguson 1992] Innes A. Ferguson. *Touring Machines: Autonomous Agents with Attitudes*. Computer, vol. 25, no. 5, pages 51–55, 1992. 36, 55, 112

[Flach 1994] Peter Flach. *Simply Logical: Intelligent Reasoning by Example*. John Wiley & Sons, Inc., New York, NY, USA, 1994. 64

[Fox & Long 2003] Maria Fox and Derek Long. *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. J. Artif. Intell. Res. (JAIR), vol. 20, pages 61–124, 2003. 126

[Franklin 1998] David Franklin. *Cooperating with people: the Intelligent Classroom*. In AAAI/IAAI, pages 555–560, 1998. v, 15, 23

[Frick 1996] Robert W. Frick. *The appropriate use of null hypothesis testing*. Psychological Methods, vol. 1, no. 4, pages 379–390, 1996. 165

[Fujii & Suda 2004] Keita Fujii and Tatsuya Suda. *Dynamic service composition using semantic information*. In ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing, pages 39–48, New York, NY, USA, 2004. ACM. v, 15, 24

[Garlan *et al.* 2002] David Garlan, Dan Siewiorek, Asim Smailagic and Peter Steenkiste. *Project Aura: Toward Distraction-Free Pervasive Computing*. IEEE Pervasive Computing, vol. 1, no. 2, pages 22–31, 2002. v, 15, 24

[Georgeff 1988] M. P. Georgeff. *Communication and interaction in multi-agent planning*. pages 200–204, 1988. 42

[Gerber *et al.* 2005] Andreas Gerber, Matthias Klusch and Marcus Schmidt. *Semantic Web Service Composition Planning with OWLS-Xplan*. In Proceedings 1st International AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, 2005. 184, 187

[Ghallab *et al.* 1998] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld and David Wilkins. *PDDL – The Planning Domain Definition Language*. Technical report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. 39, 67, 80, 123, 183

[Giersich & Kirste 2007] Martin Giersich and Thomas Kirste. *Effects of Agendas on Model-based Intention Inference of Cooperative Teams*. In Proceedings of CollaborateCom, 2007. 4

[Giersich 2010] Martin Giersich. *Real-time Intention Analysis in Teams: Concept of a Robust and Training-free Probabilistic System for Real-time Intention Analysis in Teams*. Suedwestdeutscher Verlag fuer Hochschulschriften, 2010. 80

[Gomadam *et al.* 2005] K. Gomadam, K. Verma, D. Brewer, A. Sheth and J. Miller. *Radiant: A tool for semantic annotation of Web Services*. In 4th International Semantic Web Conference, 2005. 186, 192

[Hamann & Wörn 2007] Heiko Hamann and Heinz Wörn. *Embodied Computation*. Parallel Processing Letters, vol. 17, no. 3, pages 287–298, September 2007. 46, 54

[Heider & Kirste 2002] Thomas Heider and Thomas Kirste. *Supporting Goal-Based Interaction with Dynamic Intelligent Environments*. In Proceedings of ECAI'2002, pages 596–600, 2002. 40, 144, 183, 185, 188

[Heider 2010] Thomas Heider. *Goal-based Interaction with Smart Environments: A Unified Distributed System Architecture*. Suedwestdeutscher Verlag fuer Hochschulschriften, 2010. 28, 80, 152

[Horrocks *et al.* 2004] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission, World Wide Web Consortium, 2004. 192

[IPC 2008] IPC. *Booklet of the competing planners in the sixth International Planning Competition (IPC-6)*. Sydney, Australia, 2008. 26

[Issarny *et al.* 2005] Valérie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Françoise Sailhan, Rafik Chibout, Nicole Levy and Angel Talamona. *Developing Ambient Intelligence Systems: A Solution based on Web Services*. Automated Software Engg., vol. 12, no. 1, pages 101–137, 2005. v, 15, 17, 43

[Itoh *et al.* 1999] M. Itoh, G. Abe and K. Tanaka. *Trust in and Use of Automation: Their Dependence on Occurrence Patterns of Malfunctions*. In Proc. IEEE Systems, Man, and Cybernetics Conference, pages 715–720, 1999. 146

[Jeronimo & Weast 2003] Michael Jeronimo and Jack Weast. *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003. 193

[Juric 2006] Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS*. Packt Publishing, 2nd edition, 2006. 38

[Kaelbling 1991] Leslie Pack Kaelbling. *A situated-automata approach to the design of embedded agents*. SIGART Bull., vol. 2, no. 4, pages 85–88, 1991. 48

[Karrer *et al.* 2009] Katja Karrer, Charlotte Glaser, Caroline Clemens and Carmen Bruder. *Technikaffinität erfassen - der Fragebogen TA-EG*. In Der Mensch als Mittelpunkt technischer Systeme. 8. Berliner Werkstatt Mensch-Maschine-Systeme (ZMMS Spektrum, Reihe 22, Nr. 29), pages 196–201, 2009. 160

[Kirste 2000] Thomas Kirste. *Spezifikation des Planungsassistenten*. Internal Document of the EMBASSI project, 2000. v, 15, 23

[Klein 2004] Michael Klein. *Handbuch zur DIANE Service Description*. Technical report 2004-17, Universität Karlsruhe, Faculty of Informatics, December 2004. 186, 190

[KNX 2009] *KNX Association (Official Website)*, 2009. `http://www.knx.org` (accessed November 19, 2009). 33

[Kopecký *et al.* 2006] Jacek Kopecký, Dumitru Roman, Matthew Moran and Dieter Fensel. *Semantic Web Services Grounding*. In AICT-ICIW '06, page 127, Washington, DC, USA, 2006. IEEE Computer Society. 189

[Kopecký *et al.* 2007] Jacek Kopecký, Tomas Vitvar, Carine Bournez and Joel Farrell. *SAWSDL: Semantic Annotations for WSDL and XML Schema*. IEEE Internet Computing, vol. 11, no. 6, pages 60–67, 2007. 186, 192

[Küngas 2002] Peep Küngas. *Embedding Symbolic Reasoning to Reactive Control*. In Proceedings of the Baltic Conference, BalticDB&IS 2002, pages 263–268. Institute of Cybernetics at Tallin Technical University, 2002. 36, 56

[Küster *et al.* 2007] Ulrich Küster, Birgitta König-Ries, Mirco Stern and Michael Klein. *DIANE: an integrated approach to automated service discovery, matchmaking and composition.* In WWW '07, pages 1033–1042, New York, NY, USA, 2007. ACM Press. 43, 190

[Küster *et al.* 2008] Ulrich Küster, Birgitta König-Ries and Andreas Krug. *OPOSSum - An Online Portal to Collect and Share Semantic Service Descriptions.* In Proceedings of the 5th European Semantic Web Conference (ESWC08), Poster Session, Tenerife, Canary Islands, Spain, June 2008. 194

[Laitenberger & Dreyer 1998] O. Laitenberger and H.M. Dreyer. *Evaluating the usefulness and the ease of use of a Web-based inspection data collection tool.* In Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International, pages 122–132, Nov 1998. 148

[Lausen *et al.* 2005] H. Lausen, A. Polleres and D. Roman. *Web Service Modeling Ontology (WSMO).* W3C Member Submission, 2005. 186, 189

[Lee & See 2004] John D. Lee and Katrina A. See. *Trust in automation: Designing for appropriate reliance.* Human Factors, vol. 46, pages 50–80, 2004. 142, 145, 146

[Lesser *et al.* 1999] Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan and Shelly XQ Zhang. *A Multi-Agent System for Intelligent Environment Control.* Technical report, University of Massachusetts, January 1999. v, 15, 22

[Lieberman & Espinosa 2006] Henry Lieberman and José Espinosa. *A goal-oriented interface to consumer electronics using planning and commonsense reasoning.* In IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces, pages 226–233, New York, NY, USA, 2006. ACM Press. 41, 144

[Liu *et al.* 2003] Jiming Liu, Chi Kuen Wong and Ka Keung Hui. *An Adaptive User Interface Based On Personalized Learning.* IEEE Intelligent Systems, vol. 18, no. 2, pages 52–57, 2003. 179

[Lueg 2002] Christopher Lueg. *On the Gap between Vision and Feasibility*. In Pervasive '02: Proceedings of the First International Conference on Pervasive Computing, pages 45–57, London, UK, 2002. Springer-Verlag. 152

[Maes 1989] Pattie Maes. *The Dynamics of Action Selection*. In IJCAI, pages 991–997, 1989. 109

[Maes 1990a] Pattie Maes. *How To Do The Right Thing*. Connection Science Journal, Special Issue on Hybrid Systems, vol. 1, pages 291–323, 1990. 97

[Maes 1990b] Pattie Maes. *Situated Agents Can Have Goals*. In Pattie Maes, editor, Designing Autonomous Agents, pages 49–70. MIT Press, 1990. 11, 45, 51, 103

[Marquardt & Uhrmacher 2009] Florian Marquardt and Adelinde Uhrmacher. *Creating AI Planning Domains for Smart Environments Using PDDL*. In Intelligent Interactive Assistance and Mobile Multimedia Computing, pages 263–274. Springer, 2009. 16, 25

[Martin *et al.* 2004] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, TR Payne*et al.* *OWL-S: Semantic Markup for Web Services*. W3C Member Submission, 2004. 186, 187

[Maxwell & Delaney 2003] Scott E. Maxwell and Harold D. Delaney. *Designing Experiments and Analyzing Data: A Model Comparison Perspective*. Routledge Academic, 2nd edition, 2003. 167

[Mayring 2000] Philipp Mayring. *Qualitative Content Analysis*. FQS, vol. 1, no. 2, 2000. 173

[Misker *et al.* 2005] Jan M. V. Misker, Jasper Lindenberg and Mark A. Neerincx. *Users want simple control over device selection*. In sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence, pages 129–134, New York, NY, USA, 2005. ACM. v, 15, 20

[Mokhtar *et al.* 2005] Sonia Ben Mokhtar, Nikolaos Georgantas and Valérie Issarny. *Ad Hoc Composition of User Tasks in Pervasive Computing Envi-*

ronments. In Thomas Gschwind, Uwe Aßmann and Oscar Nierstrasz, editors, Software Composition, volume 3628 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2005. v, 15, 18

[Money & Turner 2004] William Money and Arch Turner. *Application of the Technology Acceptance Model to a Knowledge Management System*. In HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 8, page 80237.2, Washington, DC, USA, 2004. IEEE Computer Society. 148

[Mozer 2005] Michael C. Mozer. *Lessons from an adaptive home*. Smart Environments: Technology, Protocols, and Applications, pages 273–298, 2005. v, 15, 20, 50

[Muir & Moray 1996] Bonnie M. Muir and Neville Moray. *Trust in automation. Part II. Experimental studies of trust and human intervention in a process control simulation*. Ergonomics, vol. 39, no. 3, pages 429–460, 1996. 142, 145, 147, 154, 170, 174

[Muir 1994] Bonnie M. Muir. *Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems*. Ergonomics, vol. 37, no. 11, pages 1905–1922, 1994. 30, 142, 150

[Müller *et al.* 1994] Jörg P. Müller, Markus Pischel and Michael Thiel. *Modeling Reactive Behaviour in Vertically Layered Agent Architectures*. In Michael Wooldridge and Nicholas R. Jennings, editors, ECAI Workshop on Agent Theories, Architectures, and Languages, volume 890 of *Lecture Notes in Computer Science*, pages 261–276. Springer, 1994. 54, 55

[Müller 1996] Jörg P. Müller. *A Cooperation Model for Autonomous Agents*. In Jörg P. Müller, Michael Wooldridge and Nicholas R. Jennings, editors, ATAL, volume 1193 of *Lecture Notes in Computer Science*, pages 245–260. Springer, 1996. 57

[Paluska *et al.* 2006] Justin Mazzola Paluska, Hubert Pham, Umar Saif, Chris Terman and Steve Ward. *Reducing Configuration Overhead with Goal-oriented Programming*. Pervasive Computing and Communications Workshops, IEEE International Conference on, vol. 0, pages 596–599, 2006. v, 15, 21

[Paolucci & Sycara 2003] Massimo Paolucci and Katia P. Sycara. *Autonomous Semantic Web Services*. IEEE Internet Computing, vol. 7, no. 5, pages 34–41, 2003. 38

[Parasuraman 1997] Raja Parasuraman. *Humans and Automation: Use, Misuse, Disuse, Abuse*. Human Factors, vol. 39, no. 2, pages 230–253, 1997. 31, 142, 145, 150, 170

[Plociennik *et al.* 2009] Christiane Plociennik, Christoph Burghardt, Florian Marquardt, Thomas Kirste and Adelinde Uhrmacher. *Modelling Device Actions in Smart Environments*. In Proceedings of International Conference on Intelligent Interactive Assistance and Mobile Multimedia Computing, Rostock, Germany, November 9-11 2009. 13

[Plociennik *et al.* 2010] Christiane Plociennik, Hartmut Wandke and Thomas Kirste. *What Influences User Acceptance of Ad-hoc Assistance Systems? – A Quantitative Study*. In Proceedings of MMS, Göttingen, Germany, February 23-25 2010. 13

[Proud *et al.* 2003] Ryan W. Proud, Jeremy J. Hart, and Richard B. Mrozinski. *Methods for Determining the Level of Autonomy to Design into a Human Spaceflight Vehicle: A Function Specific Approach*. In Proceedings of the 2003 Conference on Performance Metric for Intelligent Systems, 2003. 6

[Qiu *et al.* 2006] Lirong Qiu, Fen Lin, Changlin Wan and Zhongzhi Shi. *Semantic Web Services Composition Using AI Planning of Description Logics*. In APSCC ’06, pages 340–347, Washington, DC, USA, 2006. IEEE Computer Society. 184

[Rao *et al.* 2006] Jinghai Rao, Peep Küngas and Mihhail Matskin. *Composition of semantic web services using linear logic theorem proving*. Inf. Syst., vol. 31, no. 4, pages 340–360, 2006. 38

[Reisse & Kirste 2008a] Christiane Reisse and Thomas Kirste. *A Distributed Action Selection Mechanism for Device Cooperation in Smart Environments*.

In Proceedings of the 4th International Conference on Intelligent Environments, Seattle, USA, 2008. 12, 13

[Reisse & Kirste 2008b] Christiane Reisse and Thomas Kirste. *A distributed mechanism for device cooperation in Smart Environments*. In Advances in Pervasive Computing. Adjunct proceedings of the 6th International Conference on Pervasive Computing, pages 53–56, Sydney, Australia, May 19-22 2008. 12, 13

[Reisse *et al.* 2007] Christiane Reisse, Thomas Heider and Thomas Kirste. *A survey of Ambient Intelligence projects regarding the generation of strategies for coherent intelligent behavior*. In Proceedings of KI'2007 Workshop: Towards Ambient Intelligence: Methods for Cooperating Ensembles in Ubiquitous Environments (AIM-CU), Osnabrueck, Germany, 2007. 12

[Reisse *et al.* 2008a] Christiane Reisse, Christoph Burghardt, Florian Marquardt and Thomas Kirste. *Intelligente Umgebungen und das Semantic Web*. Information Management & Consulting, vol. 23(2), pages 28–33, May 2008. 13

[Reisse *et al.* 2008b] Christiane Reisse, Christoph Burghardt, Florian Marquardt, Thomas Kirste and Adelinde Uhrmacher. *Smart Environments Meet the Semantic Web*. In Proceedings of the 7th International ACM Conference on Mobile and Ubiquitous Multimedia, Umeå, Sweden, December 3-5 2008. 13

[Röcker *et al.* 2005] Carsten Röcker, Maddy D. Janse, Nathalie Portolan and Norbert Streitz. *User requirements for intelligent home environments: a scenario-driven approach and empirical cross-cultural study*. In sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence, pages 111–116, New York, NY, USA, 2005. ACM. 30

[Russell & Norvig 2003] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003. 70

[Russell & Norvig 2010] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 3rd edition, 2010. 39, 68

[Saif *et al.* 2003] Umar Saif, Hubert Pham, Justin Mazzola Paluska, Jason Waterman, Chris Terman and Steve Ward. *A Case for Goal-oriented Programming Semantics*. In Workshop on System Support for Ubiquitous Computing (UbiSys'03) at UbiComp 2003, Seattle, USA, Oct 2003. 41, 144

[Scholtz & Consolvo 2004] Jean Scholtz and Sunny Consolvo. *Toward a Framework for Evaluating Ubiquitous Computing Applications*. IEEE Pervasive Computing, vol. 3, no. 2, pages 82–88, 2004. 148

[Sheridan 2002] Thomas B. Sheridan. *Humans and Automation: System Design and Research Issues*. John Wiley & Sons, Inc., New York, NY, USA, 2002. 6

[Sheshagiri *et al.* 2004] Mithun Sheshagiri, Norman M. Sadeh and Fabien Gandon. *Using Semantic Web Services for Context-Aware Mobile Applications*. In Proceedings of MobiSys2004 workshop on context awareness, pages 782–796, Boston USA, 2004. 184

[Singleton 2002] Darran Singleton. *An Evolvable Approach to the Maes Action Selection Mechanism*. citeseer.ist.psu.edu/536400.html, 2002. 52

[Sirin *et al.* 2004] E. Sirin, B. Parsia, D. Wu, J. Hendler and D. Nau. *HTN planning for web service composition using SHOP*. Journal of Web Semantics, vol. 1 (4), pages 377–396, 2004. 184

[Smith 1980] Reid G. Smith. *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver.* IEEE Trans. Computers, vol. 29, no. 12, pages 1104–1113, 1980. 44

[Spiekermann 2008] Sarah Spiekermann. *User Control in Ubiquitous Computing: Design Alternatives and User Acceptance*. Shaker, Aachen, 2008. 149

[Spivey 1992] J. Michael Spivey. *The Z notation: a reference manual*. 2nd edition, 1992. 64

[Stoy 1977] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977. First edition published in 1940. 65

[Tyrrell 1993] Toby Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, 1993. 103

[Vallée *et al.* 2005] M. Vallée, F. Ramparany and L. Vercouter. *Flexible Composition of Smart Device Services*. In The 2005 International Conference on Pervasive Systems and Computing (PSC-05), Las Vegas, USA, Jun 27-30 2005. 43

[Van Gemmert & Van Galen 1997] Arend Van Gemmert and Gerard Van Galen. *Stress, neuromotor noise, and human performance: A theoretical perspective*. Journal of Experimental Psychology: Human Perception and Performance, vol. 23, no. 5, pages 1299–1313, 1997. 157

[van Orman Quine 2003] Willard van Orman Quine. *Mathematical Logic*. Harvard University Press, revised edition, 2003. First edition published in 1940. 203

[Venkatesh *et al.* 2003] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis and Fred D. Davis. *User Acceptance of Information Technology: Toward a Unified View*. MIS Quarterly, vol. 27, no. 3, 2003. 148

[Waldinger 2001] Richard J. Waldinger. *Web Agents Cooperating Deductively*. In FAABS '00: Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems-Revised Papers, pages 250–262, London, UK, 2001. Springer-Verlag. 38

[Wandke 2005] Hartmut Wandke. *Assistance in human-machine interaction: a conceptual framework and a proposal for a taxonomy*. Theoretical Issues in Ergonomics Science, vol. 6, no. 2, pages 129–155, 2005. 6, 7, 143, 151

[Weiser 1991] Mark Weiser. *The Computer for the 21st Century*. Scientific American, vol. 265, no. 3, pages 66–75, September 1991. 2, 30

[Weiss 1999] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1999. 36

[Weyns *et al.* 2008] Danny Weyns, Nelis Boucké and Tom Holvoet. *A field-based versus a protocol-based approach for adaptive task assignment*. Autonomous Agents and Multi-Agent Systems, vol. 17, no. 2, pages 288–319, 2008. 47

[Witt *et al.* 2007] H. Witt, T. Nicolai and H. Kenn. *The WUI-Toolkit: A Model-Driven UI Development Framework for Wearable User Interfaces*. In 27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07), June 2007. 178

[Wooldridge 2001] Michael J. Wooldridge. *Multi-agent systems : an introduction*. Wiley, Chichester, 2001. GBA1-Z6596 Michael Woolridge. 32, 36, 38, 45, 53, 63, 113, 114

[Wu *et al.* 2003] Dan Wu, Bijan Parsia, Evren Sirin, James A. Hendler and Dana S. Nau. *Automating DAML-S Web Services Composition Using SHOP2*. In Proceedings of the 2nd International Semantic Web Conference, pages 195–210, 2003. 184

[Yang *et al.* 2006] Yuping Yang, Fiona Mahon, M. Howard Williams and Tom Pfeifer. *Context-Aware Dynamic Personalised Service Re-composition in a Pervasive Service Environment*. In Jianhua Ma, Hai Jin, Laurence Tianruo Yang and Jeffrey J. P. Tsai, editors, UIC, volume 4159 of *Lecture Notes in Computer Science*, pages 724–735. Springer, 2006. v, 15, 25

[Ziliak & McCloskey 2004] Stephen T. Ziliak and Deirdre N. McCloskey. *Size Matters: The Standard Error of Regressions in the American Economic Review*. Journal of Socio-Economics, vol. 33, no. 5, pages 527–546, November 2004. 165

Theses

1. In environments with a complex device infrastructure it is beneficial to assist the user proactively in operating this infrastructure. In other words, it is beneficial to make such environments *smart*.
2. In smart environments, the device ensemble should generate and execute a strategy that fulfills the user's goals in the environment, while the user should be given the possibility to override the decisions of the ensemble.
3. The strategy synthesis in smart environments should be able to generate sequences of actions of moderate length.
4. The strategy synthesis in smart environments should exhibit rational behavior.
5. The strategy synthesis in smart environments should support persistent actions.
6. Smart ad-hoc smart environments do not contain a fixed device infrastructure: Devices may join and leave at run-time. Here, the strategy synthesis should be carried out spontaneously.
7. In smart ad-hoc environments, the strategy synthesis should be robust.
8. In smart ad-hoc environments, the strategy synthesis should be flexible.
9. In smart ad-hoc environments, it is desirable that the strategy synthesis is carried out in a distributed fashion.
10. If the goals of the user are known to the ensemble, it can generate an action sequence that fulfills these goals in a completely distributed fashion.
11. Neither reactive nor deliberative approaches for the strategy synthesis are optimal for smart ad-hoc environments. A hybrid approach that combines both paradigms is suited better.
12. The strategy synthesis mechanism we propose in this thesis is hybrid: It combines a reactive mechanism with a deliberative step. Hence, it profits from the advantages of both paradigms: It is architecturally simple and can be carried out completely distributed like reactive approaches, but is goal-oriented like deliberative approaches.
13. The approach is based on the assumption that each device provides declar-

ative descriptions of its actions to the ensemble. Furthermore, all devices carry out the same algorithm and select actions at run-time via communication. Hence, the approach is spontaneous, flexible, and robust, which is key in ad-hoc environments.

14. The approach does not only allow for distribution at run-time, but also at design-time: Developers can write action descriptions for devices without knowing about other developers' action descriptions.
15. As the approach is completely distributed and no device has global knowledge, the resulting action sequences are often not optimal, or, in other words, the assistance is not completely rational. Nevertheless, users accept such assistance if they perceive that it offers them an advantage compared to manual control of the environment.
16. Acceptance of an assistance system for smart ad-hoc environments depends on:
    (a) task load: Under the influence of an increased task load, people perceive the assistance system as more useful than when relaxed.
    (b) experience: Experienced users find the assistance system easier to use than those with no experience.
    (c) the behavior of the automatic assistance: When the assistance system exhibits suboptimal behavior, people perceive it as less useful than when it acts fully rational. On the other hand, the more benefit the automatic assistance offers over manual control, the more useful it is perceived even if it does not exhibit fully rational behavior.
    (d) technophilia: Moderately and very technophilic people perceive the assistance system as more useful than averagely technophilic people.
17. In smart environments, it is a challenge to make action descriptions of different devices compatible. To achieve this, the action descriptions must be written in a common language. In principle, semantic web languages are suited for this.

Note

Christiane Reiße is Christiane Plociennik's birth name.

E-mail: chre@hrz.tu-chemnitz.de