Decision-Theoretic Planning for User-Adaptive Systems: Dealing With Multiple Goals and Resource Limitations

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

vorgelegt von

Thorsten Bohnenberger

Saarbr¨ucken
Oktober 2004

Datum des Kolloquiums:
3. Dezember 2004
Dekan:
Prof. Dr. J¨org Eschmeier
Vorsitzender:
Prof. Dr. J¨org H. Siekmann
Gutachter:
1. Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster
2. Prof. Dr. Nicola Henze

Erklärung

Hiermit erkl¨are ich, dass ich die vorliegende Arbeit ohne unzul¨assige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt ¨ubernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Diese Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ¨ahnlicher Form in anderen Pr¨ufungsverfahren vorgelegt.

Saarbr¨ucken, den 17. Dezember 2004

Danksagung

Die vorliegende Arbeit entstand im Projekt READY des von der Deutschen Forschungsgemeinschaft (DFG) geförderten Sonderforschungsbereichs 378 *Ressourcenadaptive kognitive Prozesse* an der Universität des Saarlandes in Saarbrücken.

Ein besonderer Dank gilt Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster, der für die Leitung des faszinierenden Umfeldes, in dem die vorliegende Arbeit entstand, verantwortlich ist. Die Anstellung an seinem Lehrstuhl für Künstliche Intelligenz hat es mir ermöglicht, das interessante Thema der Anwendung entscheidungstheoretischer Planung für benutzeradaptive Systeme im Rahmen einer Doktorarbeit in einem interdisziplinären Umfeld zu bearbeiten. Ich danke ihm für seine zahlreichen Anregungen und das Interesse, mit dem er diese Arbeit begleitet hat.

Ebenso danke ich Prof. Dr. Anthony Jameson für die intensive Betreuung meiner Arbeit, für zahlreiche Anregungen und kreative Vorschläge, von denen die in der vorliegenden Arbeit beschriebene Forschung geprägt ist. Seiner Erfahrung verdanke ich das Heranreifen meiner eigenen wissenschaftlichen Arbeitsweise.

Ich danke Frau Prof. Dr. Nicola Henze für die spontane Bereitschaft die vorliegende Disseration als zweite Gutachterin zu beurteilen, sowie Prof. Dr. Jörg Siekmann dafür, dass er den Vorsitz des Prüfungsausschusses übernommen hat. Meine Zeit in der *Multiagentensystemgruppe* der AG Siekmann am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) stellt eine wichtige Station auf dem Weg zu meiner Promotion dar.

Für das Korrekturlesen meiner Arbeit danke ich in besonderem Maße Prof. Dr. Anthony Jameson, aber auch Dr. Steve Allen, Rainer Wasinger, Ilhan Aslan, Oliver Jacobs, Dr. Jörg Baus und Tim Schwartz, die wichtige Teile der Arbeit nochmals überprüft haben.

Allen Mitarbeitern und Studenten des Lehrstuhls für Künstliche Intelligenz an der Universität des Saarlandes gilt mein Dank für die angenehme und produktive Arbeitsatmosphäre. Insbesondere danke ich diesbezüglich meinen beiden Kollegen Dr. Frank Wittig und Tim Schwartz, die während der vergangenen vier Jahre das Büro mit mir geteilt haben.

Die erste Benutzerstudie, die in der vorliegenden Arbeit beschrieben wird, wurde unter der Mitarbeit von Prof. Dr. Andreas Butz und Dr. Antonio Krüger durchgeführt. Die Firma Eyeled GmbH hat die Infrarot-Baken für die erste Benutzerstudie zur Verfügung gestellt. Die zweite Benutzerstudie wurde selbständig von Oliver Jacobs durchgeführt. Für die Implementierung des Systems, das in der zweiten Benutzerstudie verwendet wurde, danken wir Ilhan Aslan.

Gewidmet ist diese Arbeit meinen Eltern und meinen Großeltern, die mein berufliches Interesse immer und in jeder Form unterstützt haben.

Thorsten Bohnenberger im Oktober 2004

Ja, mach nur einen Plan
Sei nur ein großes Licht!
Und mach dann noch 'nen zweiten Plan
Geh'n tun sie beide nicht.

Bertolt Brecht, Dreigroschenoper

Kurzzusammenfassung

Während eine Reihe von benutzeradaptiven Systemen existiert, die entscheidungstheoretische Methoden zum Zwecke individueller Entscheidungen verwenden, wurde die entscheidungstheoretische Planung im Kontext benutzeradaptiver Systeme bislang kaum genutzt. Die vorliegende Arbeit fokussiert die Anwendung entscheidungstheoretischer Planung in benutzeradaptiven Systemen und demonstriert, wie bei einem solchen Ansatz konkurrierende Ziele und Ressourcenbeschränkungen des Benutzers berücksichtigt werden können. Der Ansatz wird mit Beispielen aus folgenden Anwendungsgebieten illustriert: benutzeradaptive Assistenz bei der Bedienung eines technischen Geräts, benutzeradaptive Navigationsempfehlungen in einen Flughafenszenario und schließlich benutzeradaptive und ortsbezogene Einkaufsassistenz. Anhand des Einkaufsassistenten wurden in zwei Benutzerstudien Bedienbarkeitsfragen eines auf entscheidungstheoretischer Planung basierenden Systems untersucht.

Wir beschreiben, wie man in einem entscheidungstheoretischen Ansatz feste Zeitbeschränkungen, die beispielsweise in einem Flughafennavigationsszenario durch das Boarding des Fluggasts bedingt werden, berücksichtigen kann. Ausserdem schlagen wir einen auf Zielpriorisierung basierenden, hierarchischen entscheidungstheoretischen Planungsansatz vor, mit dem die Komplexität der Behandlung realistischer Probleme handhabbar gehalten wird. Weiterhin präzisieren wir den allgemeinen Arbeitsablauf (Workflow) für die Entwicklung und Anwendung von Markov Entscheidungsprozessmodellen im Kontext benutzeradaptiver Systeme, und wir beschreiben Möglichkeiten der Erweiterung eines auf entscheidungstheoretischer Planung basierenden benutzeradaptiven Systems um eine Erklärungskomponente.

Short Abstract

While there exists a number of user-adaptive systems that use decision-theoretic methods to make individual decisions, decision-theoretic planning has hardly been exploited in the context of user-adaptive systems so far. This thesis focuses on the application of decision-theoretic planning in user-adaptive systems and demonstrates how competing goals and resource limitations of the user can be considered in such an approach. The approach is illustrated with examples from the following domains: user-adaptive assistance for operating a technical device, user-adaptive navigation recommendations in an airport scenario, and finally user-adaptive and location-aware shopping assistance. With the shopping assistant, we have analyzed usability issues of a system based on decision-theoretic planning in two user studies.

We describe how hard time constraints, as they are induced, for example, by the boarding of the passenger in an airport navigation scenario, can be considered in a decision-theoretic approach. Moreover, we propose a hierarchical decision-theoretic planning approach based on goal prioriza-tion, which keeps the complexity of dealing with realistic problems tractable. Furthermore, we specify the general workflow for the development and application of Markov decision processes to be applied in user-adaptive systems, and we describe possibilities to enhance a user-adaptive system based on decision-theoretic planning by an explanation component.

Zusammenfassung

Ziel der Arbeit war die Konzeption, Entwicklung, Anwendung und Validierung entscheidungstheoretischer Planungsmethoden für Probleme im Kontext benutzeradaptiver Systeme. Insbesondere wurde dabei gezeigt, dass der Einsatz entscheidungstheoretischer Planung geeignet ist, um die aktuelle Situation des Benutzers, und damit dessen möglicherweise konkurrierende Ziele und gegebenenfalls vorhandene Ressourcenbeschränkungen, zu berücksichtigen.

Während benutzeradaptive Systeme bisweilen häufig nur die Situation zu einem bestimmten Zeitpunkt als Grundlage der Adaptionsentscheidung betrachten, erlaubt der Einsatz enscheidungstheoretischer Planung das Vorausschauen auf mehrere Interaktionsschritte in der Zukunft und damit die Berücksichtigung der Konsequenzen, die Adaptionsentscheidungen im weiteren Verlauf einer Interaktion haben können. Das bedeutet, dass das Systemverhalten nicht in Bezug auf den Nutzen eines einzelnen, nämlich des nächsten Schritts optimiert wird, sondern dass ein System in die Lage versetzt wird den erwarteten Nutzen einer Sequenz von Interaktionsschritten (Aktionen des Systems und des Benutzers), möglicherweise der vollständigen Sequenz, die zum Erreichen eines Interaktionsziels benötigt wird, zu maximieren. Wie der Begriff "erwarteter Nutzen" bereits andeutet, wird dabei die Unsicherheit bezüglich der Handlungen des Benutzers, bzw. der Ergebnisse seiner Handlungen, berücksichtigt.

Anhand verschiedener Beispiele werden entscheidungstheoretische Modelle zur Planung der Interaktion von System und Benutzer im Kontext benutzeradaptiver Systeme entwickelt. Zwei Hauptanwendungsbeispiele werden in der Arbeit betrachtet: (1) die Assistenz bei der Benutzung eines technischen Gerätes und (2) die Assistenz bei der Navigation innerhalb eines komplexen Gebäudes. Beiden Beispieldomänen ist gemeinsam, dass bei der Vorausplanung der Interaktion zwei oder mehrere konkurrierende Interaktionsziele berücksichtigt werden müssen. Darüber hinaus wird anhand dieser Anwendungsbeispiele gezeigt, wie sich ein System auf Ressourcenbeschränkungen—insbesondere die kognitive Belastung des Benutzers, dessen subjektiv empfundenen Zeitdruck, sowie objektiv vorhandene Zeitbeschränkungen—anpassen kann. In diesem Zusammenhang wird eine mögliche Variante des Zusammenspiels einer entscheidungstheoretischen Planungskomponente und einer davon unabhängigen Benutzermodellierungskomponente erläutert. In komplexeren Szenarien erlaubt diese Art des Zusammenspiels die Berücksichtigung partiell beobachtbarer Informationen (Informationen über den Benutzer, die indirekt aus der Beobachtung seines Verhaltens geschlossen werden), ohne dass im Planungsansatz die partielle Beobachtbarkeit der für die Planung relevanten Informationen explizit modelliert werden muss.

Die Arbeit gibt einen Überblick über die wichtigsten bislang verwendeten Methoden der Entscheidungsfindung in Systemen aus dem Bereich benutzeradaptiver Systeme. Die Grundlagen der entscheidungstheoretischen Planung werden zunächst anhand eines einfachen Beispiels erläutert.

Dabei wird ein einfaches entscheidungtheoretisches Modell auf der Basis empirsch ermittelter Daten entwickelt. Danach werden zwei ausf¨uhrliche Beispiele f¨ur die Entwicklung eines entscheidungstheoretischen Modells und des aus dessen Anwendung resultierenden Handlungsschemas (engl.: policy) besprochen. In diesem Zusammenhang werden zwei unterschiedliche Rollen von Zeit bei der entscheidungstheoretischen Planung betrachtet: Zum einen Zeit als reiner Kostenfaktor unter der Annahme, dass w¨ahrend der Interaktion beliebig viel Zeit verbraucht werden kann, zum anderen Zeit als eine beschr¨ankte Ressource, die durch eine feste Deadline spezifiziert ist. Die Zuordnung eines entscheidungstheoretischen Planungsproblems zu der einen oder der anderen Kategorie bez¨uglich der Rolle der Zeit entscheidet ¨uber den f¨ur den Planungsprozess betrachteten Planungshorizont.

F¨ur komplexe Planungsprobleme, d.h. f¨ur Probleme, in denen mehrere Interaktionsziele ber¨ucksichtigt werden m¨ussen, wird in dieser Arbeit ein auf Zielpriorisierung basierender, hierarchischer entscheidungstheoretischer Planungsansatz beschrieben. In einfachsten Fall unterscheidet dieser Ansatz zwischen obligatorischen und optionalen (prim¨aren und sekund¨aren) Zielen und ber¨ucksichtigt eine feste Deadline f¨ur das Erreichen des ¨ubergeordneten Ziels. Das durch die gesteigerte Anzahl betrachteter Interaktionsziele entstehende Komplexit¨atsproblem wird dabei insofern vereinfacht, als dass durch einen Planungsprozess auf ¨ubergeordneter Ebene eine Aufteilung von Sekund¨arzielen auf Prim¨arziele erreicht wird, so dass in den Planungsprozessen auf untergeordneter Ebene jeweils lediglich Teilmengen der Gesamtmenge an Sekund¨arzielen ber¨ucksichtigt werden m¨ussen.

Die aus der Entwicklung entscheidungstheoretischer Modelle und der Anwendung der aus den entsprechenden Planungsprozessen resultierenden Handlungsschemata gewonnen Erkenntnisse werden in der Beschreibung eines allgemeinen Arbeitsablaufs (Workflow) zusammengefasst. Dieser Workflow unterscheidet die Entwicklung und Anwendung entscheidungstheoretischer Modelle f¨ur einfache und komplexere Planungprobleme. F¨ur komplexere Planungsprobleme wird der zuvor entwicklete Ansatz der zielpriorisierten, hierarchischen Dekomposition zugrunde gelegt und verallgemeinert. Die Beschreibung des Workflows kann als ein erster Schritt auf dem Weg zu einem allgemeinen Werkzeug f¨ur die Entwicklung entscheidungstheoretischer Planungsmodule f¨ur benutzeradaptive Systeme betrachtet werden.

Die Arbeit berichtet ¨uber zwei Studien zur Untersuchung der Akzeptanz des entscheidungstheoretischen Plannungsansatzes im Kontext eines mobilen Einkaufsassistenten. Die erste Studie wurde in einem fingierten Einkaufszentrum in einem Informatikgeb¨aude der Universit¨at des Saarlandes unter Verwendung von Infrarot-Lokalisierungstechnologie durchgef¨uhrt. Die zweite Studie, in der der Versuchsleiter die Infrarot-Lokalisierungstechnologie simulierte, fand im Saarparkcenter in der Stadt Neunkirchen statt. Beide Studien legen nahe, dass der entscheidungstheoretische Planungsansatz eine ad¨aquate Methode der Entscheidungsfindung f¨ur Systeme in der Art des mobilen Einkaufsassistenten darstellt. In einer weiteren Studie wurden M¨oglichkeiten der Entwicklung einer Erkl¨arungskomponente f¨ur entscheidungstheoretische Planer im Kontext mobiler Einkausassistenten untersucht. Die in der Arbeit vorgeschlagenen Strategien zur Realisierung einer solchen Erkl¨arungskomponente stellen einen weiteren Beitrag zur Steigerung der Bedienbarkeit und Benutzerfreundlichkeit auf entscheidungstheoretischer Planung basierender benutzeradaptiver Systeme dar.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation: An Example from Everyday Life

"Good evening. I'd like to go from Trier to Ludwigshafen. What are the next train connections, please?". The young lady at the ticket desk feeds her computer with some data. Then she says: "There's a connection at 7:48pm. You need to change trains in Saarbr¨ucken and Kaiserslautern. Arrival in Ludwigshafen is at 11:38pm. Another train leaves at 8:16pm. You need to change trains in Koblenz and Mannheim. Arrival in Ludwigshafen is at 11:40pm."

"Uh, it is nearly a quarter to eight. I don't want to hurry. I think, I take the second train and go to the newsagent first to buy a magazine for the journey. The arrival times are about the same, anyway. Thank you very much for your assistance!"

The two connections differ only marginally in journey and arrival time. At first sight, they seem to be more or less equally well suited for the trip to Ludwigshafen. Our traveler, let us call him T, has compared two plans consisting of three sequential actions. *Plan A:* go from Trier to Saarbr¨ucken, go from Saarbr¨ucken to Kaiserslautern, and go from Kaiserslautern to Ludwigshafen. *Plan B:* go from Trier to Koblenz, go from Koblenz to Mannheim, and go from Mannheim to Ludwigshafen. If everything works out well, there is no reason for T to favour one decision over the other. Unfortunately, railways are often not very reliable. Figure 1.1 shows the regular timetables for both plans as well as the timetables for fallback strategies if T missed any of the trains.

If T missed a train following *plan A*, no matter if in Saarbr¨ucken or Kaiserslautern, then there is another one that gets T to Ludwigshafen at 0:57am. That is, wherever T misses a train on this route, he would reach Ludwigshafen with a delay of at most about 1:20h compared to the regular arrival time. Now we reconsider *plan B*.

If T missed the train from Koblenz to Mannheim, he can choose only a connection via Frankfurt airport. From there, he will take the train via Mannheim reaching Ludwigshafen at 4:37am. This is nearly 5 hours later than the regular arrival time. Things get even slightly worse if T additionally missed the train at Frankfurt airport. In this case, the next train would get T to Ludwigshafen via Bruchsal and Mannheim only at 5:12am—about 5:30h later than the regular arrival time.

Figure 1.2 shows a graphical representation of best, bad, and worst cases for the two routes. The information was retrieved from the web server[1] of German Railways (Deutsche Bahn AG) for Monday and Tuesday, September 15 and 16. Considering what could go wrong following each

[1] Times retrieved from the schedule valid from 15.06. through 13.12.2003 at `http://www.bahn.de/`.

Plan A

Station	Time
Trier	7:48pm
Saarbrücken	8:55pm
Saarbrücken	9:02pm
Kaiserslautern	10:02pm
Kaiserslauter	10:31pm
Ludwigshafen	11:28pm

Station	Time
Saarbrücken	11:02pm
Ludwigshafen	0:57am

Station	Time
Kaiserslautern	0:00am
Ludwigshafen	0:57am

Plan B

Station	Time
Trier	8:12pm
Koblenz	9:37pm
Koblenz	9:48pm
Mannheim	11:20pm
Mannheim	11:35pm
Ludwigshafen	11:40pm

Station	Time
Koblenz	10.48pm
Frankfurt airport	0:00am
Frankfurt airport	0:27am
Mannheim	1:05am
Mannheim	4:33am
Ludwigshafen	4:37am

Station	Time
Frankfurt airport	2:20am
Bruchsal	3:32am
Bruchsal	4:31am
Mannheim	5:01am
Mannheim	5:08am
Ludwigshafen	5:12am

regular

fall back strategies

***Figure 1.1**: Timetables to get from Trier to Ludwigshafen by train.*
(The decision about the connection is often mainly based on a best case analysis. Fallback strategies—in this example possible connections after missing a train at some point—are usually not considered.)

of the two plans might well make T change his mind and decide not to buy a magazine first, but instead hurry a bit to get the 7:48pm train from Trier to Ludwigshafen via Saarbr ̈ucken.

The example illustrates that it is sometimes worthwhile to look ahead and to analyze the future consequences of a decision. In fact, a more detailed analysis can be made. Taking into account how likely it is at each station that T will miss a train (e.g., based on statistics about train delays or based on an estimation of the likelihood to miss a train considering the time available to change trains) would enable T to assess the two alternatives more precisely. T's decision will probably depend also on another question: How important is it for him to be well rested the next day? If he is not working the next day, he might take the risk of not getting to sleep during the night, as he estimates the likelihood for that case to be rather low, anyway. Whereas, if he has a business meeting the next morning at 9am, he might rather want to exclude entirely that he will not get to sleep before 6am in the morning.

The kind of planning and looking ahead that we just described in the context of a railway information service can be helpful and valuable in many other domains. It is called *decision-theoretic planning (DTP)* and was originally developed in operations research to make decisions about business processes (Bellman, 1957; Howard, 1960; Astr ̈om, 1965). Beside its application in economics, decision-theoretic planning was introduced to the field of *artificial intelligence* during

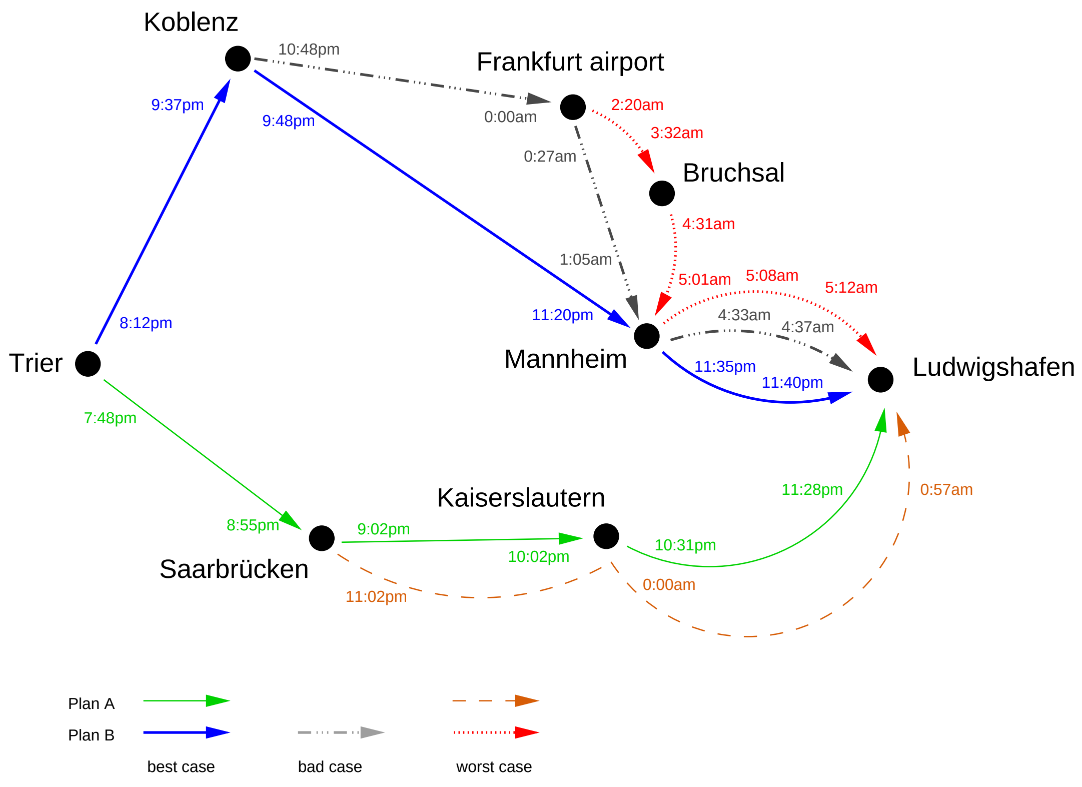


***Figure 1.2**: Graphical representation of relevant connections from Trier to Ludwigshafen.*
(If everything works out well, plan A and plan B are equally well suited to get from Trier to Ludwigshafen. But in the worst case plan B turns out to be very unfavorable.)

the last decade, primarily to tackle the difficult problem of robot control (e.g., Dean, Kaelbling, Kirman, & Nicholson, 1995; Kaelbling, Littman, & Cassandra, 1998; Boutilier, Dean, & Hanks, 1999).

This thesis is driven by the belief that decision-theoretic planning can contribute a lot to another sub-area of artificial intelligence: the area of *user-adaptive systems*. Computer users are frequently complaining about the difficulty in understanding the user interface of systems and programs.

Sometimes the problem is simply that a system's graphical user interface (GUI) makes it awkward for people to interact with it. In this situation the problems can easily be overcome by re-working the GUI's design.

More challenging problems are those that become apparent only during the interaction of the user with a system. Some systems only work well for some users and in some situations. Research to date has focused on methodologies and techniques for obtaining information about the user and the situation, and then adapting system behavior as a result of this information. For example, web pages that adapt the presentation of information to the current user. However, most intelligent systems only adapt to the current situation and fail to even consider how the situation might evolve during the course of interaction. If a system were capable of planning ahead, then it would be able to steer the interaction with the user into favorable directions and thereby avoid dead ends early during the interaction, instead of waiting until the user faces them directly. We want to enable systems to analyze potential courses of the interaction and to decide at each stage of the interaction in the way that is most beneficial for the user.

In the following, we will briefly introduce the closely related areas of *intelligent user interfaces* and *user-adaptive systems*. The two areas overlap, and it is often difficult to decide if a particular system belongs more to one or the other category. Therefore, many of the conclusions concerning the application of decision-theoretic planning for user-adaptive systems apply also for the application of decision-theoretic planning for intelligent user interfaces.

In the section about user-adaptive systems, we will highlight the concepts of *resources* and *resource-adaptivity*. These concepts provide the background for the READY project, in which the research described in this thesis was carried out.

1.2 Intelligent User Interfaces

Maybury and Wahlster (1998) define *intelligent user interfaces (IUIs)* as human-machine interfaces that aim to improve the efficiency, effectiveness, and naturalness of human-machine interaction. To realize these properties, they employ explicit models of the user, the domain tasks, the discourse of the interaction, and the available media.

Intelligent user interfaces include systems that automatically process coordinated language, gesture and gaze input, and that provide multi-modal output. Some perform automatic content selection, media allocation, media realization and layout. There are systems that can automatically design graphics from structured data or coordinate the layout of multimedia content in time and space. Intelligent user interfaces often come as anthropomorphic agents, such as assistants or mediators. Behind the scenes, intelligent user interfaces often exploit models of the discourse and user, build or maintained during interactive sessions.

Figure 1.3 illustrates in what parts of the general architecture of an intelligent user interface (as it is defined by Maybury & Wahlster, 1998) decision-theoretic planning may be involved[2]. According to this architecture, decision-theoretic planning can be employed as part of the *interaction management*. We will show in Chapters 4, 5, and 7 how decision-theoretic planning can be used for *discourse modeling*, plan generation, and *presentation design* and how it has an effect on *media design* and *output rendering*.

How intelligent the behavior of a computer system appears to its user is a function of how appropriately the system chooses its actions. The more trivial a problem is, the more a user should expect the system to act more or less like the user would do. Of course, a problem that appears trivial to a human cannot necessarily be (in fact is often not) solved trivially by a computer system—researchers in AI have often experienced this phenomenon. The more complex a problem is, the more often a system might be expected to make a decision that is unexpected by the user. For some problems (of high complexity) the user will be unable to make the right (or at least a reasonable) decision at all.

How does the railway information service example translate to the field of intelligent user interfaces? We discuss an intelligent user interface example that involves a sequential decision

[2]In principle, decision-theoretic planning can also be involved in input processing and plan recognition. Consider, for example, a situation with two persons both using assistance systems based on decision-theoretic planning, where the system of one person tries to anticipate the actions of the other, for example, to arrange a meeting.

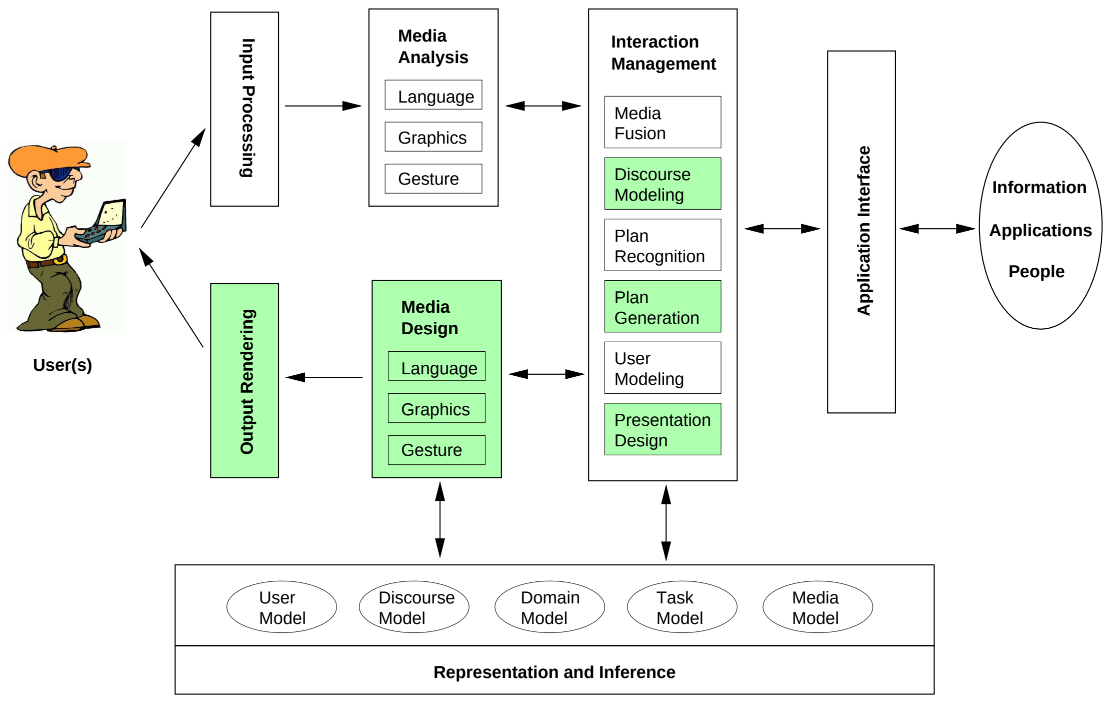


***Figure 1.3**: General architecture of an intelligent user interface.*
(Maybury and Wahlster (1998) describe the general architecture of an intelligent user interface. In this architecture, a decision-theoretic planner is considered as part of the interaction management, and it has an effect on the **Media Design** *and* **Output Rendering***.)*

making similar to the decision making in the railway information service example to answer this question: the interaction of a customer C with an automatic teller machine. The dialog between a customer and an ATM usually follows a rigid structure. The system starts by asking the customer to enter his PIN and then to select a service. Assume C most of the time uses the ATM to withdraw some cash, and C has the habit to frequently withdraw an uneven amount, say \$35. If the ATM could access information about the way C enters his PIN (e.g., rather quick or rather slow, with or without delay after entering the bank card, with or without mistake) or about C's physical state, for example, from biosensors in the keyboard (e.g., to measure skin conductivity), or from an eye tracker (e.g., to measure the pupil diameter), then it might be able to infer if C is in a hurry. Maybe C is on the way to the cinema, and the movie starts shortly. In that case, C might appreciate if the ATM offers him to withdraw the amount of \$35 immediately after he has entered his PIN. This service would save C two dialog steps because he does not need to select a service first and then enter the amount. Saving two dialog steps probably means saving less than thirty seconds. But the subjective value of saving two dialog steps might well be more than just saving some seconds (see also Smyth & Cotter, 2002, who analyze the value of personalized navigation techniques for WAP portals and information services in general). Rather, the added value achieved through the flexible dialog is reflected by the enhanced customer convenience. Figure 1.4 shows the two alternative interaction sequences (in part).

Tailoring the dialog exactly to the user's current needs is not as easy as the example might suggest at first sight. Maybe, sometimes C just wants to check his account balance, sometimes the

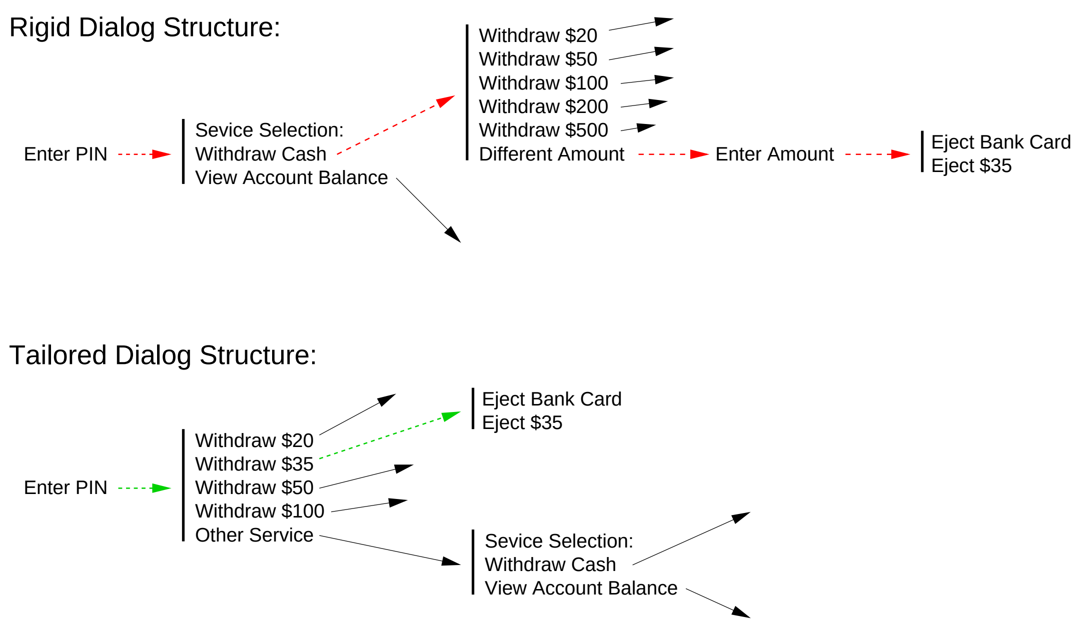


***Figure 1.4**: Two dialogs with an ATM to withdraw $35.*
(Dashed path: Following the rigid dialog structure takes four steps to withdraw $35. Green path: The user tailored dialog structure allows to withdraw $35 in only two steps.)

amount of money he wants to withdraw is a bit higher or lower than usually. If the ATM makes a wrong guess in the beginning, the result might be that C feels disturbed or irritated by the system behavior[3]. Maybe the dialog will even require an additional step compared to following the rigid strategy. A user-friendly ATM has to account for this possibility. It should be able to trade off alternatives precisely based on the likelihood of their results and the values that the results have for C. Decision-theoretic planning provides the formal methods to do so.

The example illustrates how the ATM can adapt to the customer's situation. Adapting to the customer's situation means in this case to adapt to the customer's time pressure. Time can be considered as a resource that is available to the user of a system. In this thesis, we focus on a class of systems closely related to intelligent user interfaces: *user-adaptive systems*. Some user-adaptive system take in particular the user's resource limitations into account. Such systems are sometimes explicitly labeled with the term *resource-adaptive systems*.

[3]In our previous work, we have not yet explicitly considered how the utility of an adapted system behavior compares to a behavior the customer is used to, but it could be done in principle. It involves estimating the cost to the user of adapting to a new (adapted but unexpected) system behavior.

1.3 User-Adaptive Systems

Jameson (2003) summarizes all systems that perform an adaptation to the individual user in some nontrivial way under the term *user-adaptive systems*. Depending on their form and function, such systems have been given labels ranging from *adaptive interfaces* through *user modeling systems* to *software agents* or *intelligent agents*; the broader term *personalization* became popular in the 1990s, especially in connection with commercially deployed user-adaptive systems.

Figure 1.5 surveys the general schema for the processing in a user-adaptive system given by Jameson (2003) and indicates the processing of a user-adaptive system based on decision-theoretic planning. Central to the general schema is a user model for the given task. In an approach based on decision-theoretic planning, all relevant information about the user is represented in a Markov decision process[4]. The Markov decision process comprises all relevant possible courses of the interaction between the system and the user and incorporates the parameters that specify the possible actions of the user. On the basis of the Markov decision process, the user-adaptive system can compute an interaction policy that determines the system's instructions or recommendations for the user.

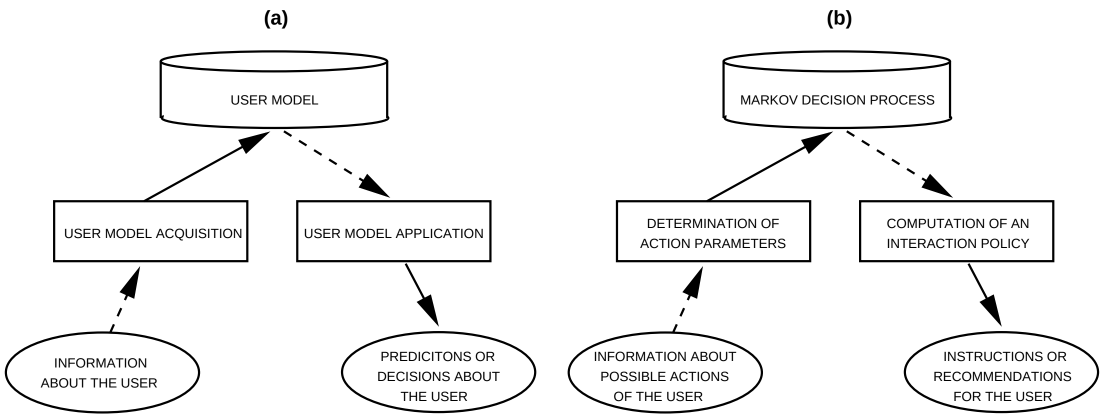

(a)
USER MODEL
USER MODEL ACQUISITION
USER MODEL APPLICATION
INFORMATION ABOUT THE USER
PREDICITONS OR DECISIONS ABOUT THE USER
(b)
MARKOV DECISION PROCESS
DETERMINATION OF ACTION PARAMETERS
COMPUTATION OF AN INTERACTION POLICY
INFORMATION ABOUT POSSIBLE ACTIONS OF THE USER
INSTRUCTIONS OR RECOMMENDATIONS FOR THE USER

***Figure 1.5**: Schema for the processing (a) in a user-adaptive system in general and (b) in a DTP-based user-adaptive system in particular.*
(Ovals: input or output; rectangles: processing methods; cylinder: stored information. Dashed arrows: use of information; solid arrows: production of results.)

1.3.1 Resources and Resource-Adaptivity

In habitual language use, the meaning of the term *resource* includes physical objects, human abilities and skills, knowledge, information, energy, time, etc. Some resources can be consumed, while others can be used arbitrarily often. Some resources can be portioned, while others can reasonably be used only as a whole. In the context of interdisciplinary research in the collaborative research program 378 on resource-adaptive cognitive processes[5]—started at Saarland University in 1996—Wahlster and Tack (1997) define the following resource concept: A cognitive agent A

[4]The Markov decision process constitutes the mathematical framework of decision-theoretic planning. We will introduce Markov decision processes in Section 3.1.
[5]Information on the SFB378 can be found at `http://www.coli.uni-sb.de/sfb378/`.

solves a given task T in a situation S using a resource R, whereby resource-dependent variations of result quality and the used processes can be observed for $A(T, S, R)$. They classify three types of resource sensitive processes. A slightly modified classification is described by Jameson and Buchholz (1998). We adopt the latter classification and illustrate it with an example of a system guiding a user through the airport on the way to his gate. The "agent" here is rather the system-user combination than the system alone, as the resource limitations are those of the user, while the behavior is determined mainly by the system.

Jameson and Buchholz (1998) divide *resource-adaptive* processes into (1) *resource-adapting*, (2) *resource-dependent*, and (3) *resource-adapted* processes. Resource-adaptivity is the superordinate concept: A process can generally be called resource-adaptive if the way in which it solves a task depends on the restrictions of the relevant resources. A system guiding a user to his gate acts in a resource-adapted way, for example, if the system recommends fixed routes, whose basic definitions take resource restrictions, such as the user's working memory capacity, implicitly into account. The system acts in a resource-dependent way if the quality of the navigation recommendations changes with different levels of working memory capacity although the system has not modeled this resource explicitly. If the user cannot follow a recommendation correctly, the system might just give another recommendation that is simpler, for example, a recommendation of which the system designers assumed that it would be easier to follow. Eventually, the system acts in a resource-adapting way if it explicitly models a resource, for example, the user's working memory capacity, and considers it in the generation of the navigation recommendations. This behavior represents the strongest notion of a resource-adaptive process.

1.3.2 READY: REsource-Adaptive Dialog sYstem

There are plenty of ways for a user to interact with a system. Pure command line interfaces from the early beginnings of desktop computers have quickly been replaced by the windows, icons, menus, and pointing (WIMP) paradigm. Mouse and keyboard are familiar to every user of a desktop PC. The spread of mobile devices has established pens as pointing devices. Some systems incorporate spoken language, and many projects pay attention to multi-modal interaction. No matter what form of input is used, there is always some information in addition to the content of the user's input to a system: symptoms that allow to make (at least vague) inferences about the user's current situation. This additional information is usually neglected and in fact is often not relevant for the application at hand. For example, if someone types some text in a command line, it is usually not important if he types it quick or slow, or how often he makes corrections in the text. Yet, the information could be used, for example, to assess if the user is a novice, an intermediate, or an advanced computer user. READY uses this kind of information (symptoms in the user's interaction with the system, for example, speech symptoms or symptoms in the usage of a pen on a PDA display) to reason about the user's cognitive load or time pressure.

The project READY (Jameson, Sch¨afer, Weis, Berthold, & Weyrath, 1999) started in 1996 as part of the collaborative research program 378 on resource-adaptive cognitive processes. From its beginnings, the research in the project was driven by two questions: (1) How can a system recognize resource limitations of a dialog partner, and (2) how can the system adapt its behavior to recognized resource limitations? The prototype developed during the first research period was based on a car breakdown scenario. The system assists car drivers in the event of a breakdown via the phone. The prototype uses dynamic Bayesian networks to reason about the user's resource limitations and influence diagrams for the decision making. These methods are especially well suited for the integration of unreliable evidence, incremental use of sparse evidence, and explicit reasoning about the ways in which the user's resource limitations change during the interaction.

This thesis summarizes the research in the READY project that deals with adaptation processes based on planning an interaction several steps ahead. Most examples throughout the thesis are placed in an airport scenario, which was jointly used by several projects of the SFB 378 during the 2nd and 3rd phase of the research program. There was a particularly tight cooperation with the partner project REAL; both projects are hosted by the AI lab at Saarland University. Two examples of services of an envisaged mobile airport assistance system will repeatedly be used in the upcoming chapters to illustrate the advantages of planning the interaction between a system and its user ahead: (1) The system shows the user round in an airport terminal. Thereby, it considers secondary desires of the user (e.g., buying a snack or finding a rest room) and eventually gets the user to the gate in time. (2) The system interactively explains to the user how to operate a technical device (e.g., a credit card phone or an automated check-in counter).

Both services have in common the facts that the outcome of the user's reaction to the system's recommendations is uncertain (e.g., the user buys a snack or not, the user operates the credit card phone correctly or not) and that the result of the whole interaction (the sequence of the system and the user's actions) cannot be optimized by optimizing individual decisions.

1.4 Planning Interactions

Planning the behavior of a user-adaptive system always means building a plan that considers the actions of two agents: those of the user and those of the system. The dialogic nature of interaction planning and its implication for the planning approach is the subject of this section.

Usually the examples employed in decision-theoretic planning literature apply to situations with a single agent only (e.g., robot navigation). Although there are usually two cooperative agents involved in human computer interaction, the problem of planning the interaction between a user and a system is usually not considered as a multi-agent planning problem[6]. The main reason for the single-agent perspective is probably the fact that one agent, the user, takes a rather passive part in the interaction in the sense that he merely reacts to the system's actions. The user does not determine his actions according to a joint plan, and to at least gain access to the user's plan, the system has to explicitly employ plan recognition methods. Yet, what might have the appearance of a two-agent planning problem at first sight can in fact easily be mapped to a single-agent planning problem. Consider the example in which the system gives a recommendation and the user tries to understand and to follow the recommendation. One way to capture this circumstance is to mask the user's actions as system actions. The user's action of following the recommendation can be masked as "the system waits for the user to follow the recommendation". This way, the user's action is described from the system's perspective although important parameters of the action (its cost, the likelihoods of its outcomes) describe properties that are related to the user.

An alternative approach, which emphasizes the passiveness of the user in the planning process even more, could be to treat the user as a part of the system's uncertain and dynamically changing environment. However, this approach degrades the role of the user too much, and makes the user's actions appear less important to the planning process than they actually are.

[6]We assume throughout the thesis that the user is cooperative. If the user tries to fool the system in some way, then there is usually something wrong with the system. In that case, that is, for non-cooperative situations, game theoretical analyses might have to be taken into consideration.

Keeping these considerations in mind will help to understand the modeling of the planning processes described in chapters 4, 5 and 7. In the following, we will take a closer look at two further aspects that are particularly important for planning the interaction of a system and a user: (1) What can the system know about the state of the user and the interaction, respectively? (2) What is the role of time in the planning process?

1.4.1 Feedback: Degrees of Observability

The selection of an appropriate planning approach for a planning problem depends on the answer to the following question: Does the agent executing the plan perceive the results of its actions, and how reliable does it perceive them? Or put in other words, how accessible are changes of the world to the plan executor at runtime?

The purpose of planning an interaction ahead of time is to have a good (or optimal with regard to some optimality criterion) strategy at hand for whatever situation might occur during the course of the interaction. Starting from this fundamental idea, those scenarios in which we can assume that the system has some means to gain some knowledge about the current situation at runtime are the most interesting ones for our studies. In some cases, a system can perceive the current situation reliably, in other cases it cannot. How good the assessment of the situation is, depends on the amount and quality of information that is available to the system. Some examples are:

- A tutoring system must know about the student's state of knowledge, that is, how much of the subject matter the student has already understood. The system can ask the student questions to find out about the student's state of knowledge. In order to take the student's current receptiveness or affective state into consideration, the system can analyze the user's input behavior, for example, it can measure how long the student takes to read an exercise or to answer a question.

- A system that gives repair instructions via the phone must find out about the state of the broken car as well as the driver's standard of technical knowledge and maybe his affective state to give reasonable recommendations. The driver's affective state can be inferred from symptoms in his speech, his standard of technical knowledge through an analysis of his vocabulary. The recognition of his affective state and standard of technical knowledge can be used for the decision about what questions to ask about the state of the car and how to phrase them.

- A system that gives egocentric (from the perspective of the user) navigation recommendations has to know what the user's current location and orientation are. Dependent on the underlying technology, the location can sometimes be sensed precisely, whereas sometimes the system will have only rough (noisy) localization information available. The user can also be involved in the localization process. The same accounts for information about the user's orientation.

- A shopping assistant, that guides a user to a number of stores to buy items from a shopping list, needs to know which items have already been bought, what the user's current location is, and which stores the user has already visited. Some of this information can be sensed automatically in an instrumented environment, some might be given by the user explicitly.

How much of this information can be considered to be reliable is crucial for the selection of the planning process that can be applied to the problem. If no information about the changes of the situation is accessible to the system, the best the system can come up with is a *plan*: a sequence of actions, which is worked off independently of the state of the world. Applying a plan implicitly assumes that actions always have the desired and unambiguous outcome, that is, that the model employed for the planning process reflects the considered fraction of the world well enough, such that nothing else can happen other than is assumed in the model. In many real world problems, this assumption does not hold, as the railway information service example at the very beginning of this chapter has already illustrated.

On the other end of the spectrum, there is the assumption that the relevant fraction of the world can be perceived perfectly. In this case, the planning process is said to be *fully observable*. The result of such a planning process is not called a plan anymore, but a *policy*: a table in which the system can look up what action to take for every possible situation. For example, the route planning problem of a GPS-based car navigation system can be considered to be fully observable if one considers the precision of GPS-based localization to be sufficient for car navigation. Chapter 3 will survey in detail the elements of *fully observable Markov decision processes (FOMDP)*, which we use to model the fully observable planning problems in chapters 4, 5, and 7.

Some processes are neither unobservable nor fully observable. Sometimes a system makes *observations* that do not allow the system to determine the current state of the world precisely, but give the system an estimation of the likelihood that the world is in any of the potential world states. For example, the user of the telephone repair service mentioned above tells the system about some steam rising from the engine compartment and a red lamp that flashes on the dash board. These symptoms do not tell the system exactly what the status of the engine is. However, the system can estimate, for example, that it is more likely that there is not enough water in the cooling unit than that the temperature sensor is broken. A planning process that takes the uncertainty about the actual world state into consideration is said to be *partially observable*.

The observability classification is not inherent to a property or a sensor type used to determine a property. For example, a route planning process based on GPS information for car navigation might be considered to be fully observable, whereas a route planning process based on GPS information for pedestrian navigation might be considered to be only partially observable. The system designer has to answer the question: Is it necessary to account for some uncertainty about the state of the world? For a car navigation system it might be sufficient to know that the user is at most 10 meters away from a crossing—the route planning process can be considered to be fully observable. The same localization technique might be insufficiently precise for a pedestrian navigation system—in this case, the route planning process might rather be considered to be partially observable.

Most state knowledge (information about the situation at hand) that will be relevant in the scenarios that we consider in chapters 4, 5 and 7 can be perceived with sufficient accuracy. Therefore, we can focus on fully observable planning processes in the best part of this thesis. We will, however, hint to scenarios in which the modeling as *partially observable Markov decision process (POMDP)* promises some extra value. While the complexity to represent FOMDPs allows at least the modeling of a certain range of problems with practical relevance (which can be extended by appropriate approximation methods as we apply them in chapter 5), POMDPs suffer even more from what is often called the MDP curse of dimensionality. There are hardly any practically relevant partially observable planning processes that can be represented as POMDPs.

1.4.2 Different Roles of Time

We distinguish two different roles of time in a planning process: (1) time as a pure *cost factor* for the interaction of a system with its user, that is, in principle as an unrestricted resource, and (2) time restricted by a *fixed deadline* specifying the end of an interaction (a latest point in time at which the interaction has to be finished), that is, time as a restricted resource.

Time as unrestricted resource (time as a pure cost factor) An interaction can be optimized with regard to the interaction duration. In many applications such optimization will require a *trade-off* between the interaction duration and some other performance criterion. Some examples:

- A tutoring system might claim to convey as much content of the curriculum in the shortest possible time; the lessons must be comprehensible enough for the student being able to follow, but they must not be unnecessarily verbose such that it bores the student;
- a system giving assistance for the driver of a broken down car might attempt to derive a good assessment of the situation and in order to being able to give good recommendations after the shortest possible period of interviewing;
- a system giving instructions on how to operate a technical device might attempt to instruct the user in a way that the user can operate the technical device correctly as fast as possible. At the same time, the system might try to minimize the probability that the user operates the device incorrectly.

In all these examples, optimizing the result of the planning process means: at the same time minimizing the required time *and* minimizing or maximizing some other variable. Time is not a restricted resource in this approach. The planned sequence of actions can in principle take as much time as required, as long as the increased duration of the interaction is justified by a commensurable increase of output quality. A slightly better result of the planned process might not justify the immoderate timely extention of the sequence of actions (e.g., reflected by the number of actions required) to achieve the result. We will see in Chapters 3 and 4 that decision-theoretic planning provides a mathematical framework that suites these requirements well. It allows to trade-off the costs of required actions against the value of the achieved result.

Time as restricted resource (time restricted by a fixed deadline) Some applications require that the interaction of a system with its user takes at most some predefined amount of time. Some typical interactions with a fixed deadline are the following:

- A shopping assistant helps a shopper to find as many of the items on the shopping list as possible before the shopping mall closes;
- a personal assistant for operating a railway ticket machine tries to finish the interaction early enough, such that the traveler can get to his platform in time;
- an airport assistance system helps a passenger to navigate within an airport building. It tries to satisfy as many of his desires (e.g., buying a snack or finding a rest room) as possible, but in particular, it cares for the passenger not to miss his flight. The boarding time determines a fixed deadline in this case.

Decision-theoretic planning allows to consider a fixed deadline in terms of a *finite-horizon* planning problem. finite-horizon planning allows to determine an optimal strategy for a fixed amount of process steps or *stages*. In the basic decision-theoretic planning framework, performing one action corresponds to proceeding one stage in the process to be planned. This notion has to be modified to match finite-horizon planning for a process that must be finished before a fixed deadline. If the costs of actions can be represented by an integer amount of time and if stages in the planning process correspond to time units, finite-horizon planning suits the requirements. Chapter 3 will explain the technique to deal with fixed deadlines formally.

Note that there is a difference between finishing the planning of a process before a deadline (the planning itself must not take longer than a certain amount of time) and planning a process in a way that the planned process is finished before a deadline. The technique that we described in the last paragraph applies to the latter case. The first case, the problem of ensuring that the planning process itself is finished after some period of time, is related to the area of anytime algorithms (see e.g., Zilberstein, 1996). This issue represents an interesting area of its own, but it is not substantial to the objectives of this thesis.

1.5 Objectives

Many user-adaptive systems base adaptation decisions solely on an analysis of the situation at hand. The overall idea of the research presented in this thesis is to provide user-adaptive systems with a means to consider future consequences of the decision on how to act in and adapt to the situation at hand. Decision-theoretic planning has successfully been applied, for example, to robot navigation problems. Just like a smart robot that sometimes decides for the fastest route and sometimes considers a deviation to avoid a dangerous, unfavorable situation, a user-adaptive system should sometimes plan its interaction with the user several steps ahead to avoid unfavorable situations for the user. We present an approach based on decision-theoretic planning, which is general enough to be applied to a broad range of domains. We group the questions that drove the research presented in this thesis into three topics: (1) General issues concerning decision-theoretic models for the planning in user-adaptive systems; (2) usability issues of user-adaptive systems based on decision-theoretic planning; and (3) special requirements of specific types of application.

General issues concerning decision-theoretic models for the planning in user-adaptive systems Decision-theoretic planning allows a system to "think out" an interaction, to compare different courses of interaction, to steer an interaction into a direction that is favorable for the user, and to avoid pitfalls that the user would otherwise be confronted with. The development of decision-theoretic models that allow to exploit the described advantages of decision-theoretic planning in the context of user-adaptive systems brings up the following questions:

- *Can we formulate general guidelines for the development of decision-theoretic models for user-adaptive systems?*

  The development and application of decision-theoretic models for various user-adaptive systems should exhibit some similarities that may be captured in a general workflow description. We will study several examples throughout this thesis, which will help to elicit the individual steps of the general development and application workflow, which must be applicable to problems of different complexity.

- *How can the parameters required for decision-theoretic planning be determined?*

  Decision-theoretic planning is based on quantitative measures that describe the available actions. In particular, these parameters are the costs of actions and the probabilities of their outcomes. We will explore appropriate ways to determine the parameters required by the models for decision-theoretic planning.

- *How can a system trade off multiple objectives?*

  One advantage of decision-theoretic methods in general is their ability to handle trade-offs between competing goals quantitatively. However, it is not obvious how to handle trade-offs between multiple goals in the context of planning for user-adaptive systems.

- *How can a fixed deadline for reaching a goal with a user-adaptive system be dealt with?*

  While planning problems with a finite planning horizon are in general simpler to solve than those with an infinite planning horizon, a fixed deadline imposes some extra effort when modeling and dissolving a decision-theoretic planning problem with non-uniform action durations. Some of the user-adaptive systems considered in this thesis require to take fixed deadlines under such conditions into account.

- *How can a reasonable look-ahead in a complex domain be accomplished?*

  More complex instances of planning problems in the context of user-adaptive systems considered in this thesis may require specific methods to make decision-theoretic planning actually applicable. We will explore existing approaches to deal with complex decision-theoretic planning problems and if necessary enhance a promising approach according to the requirements of the corresponding domain.

Usability issues of user-adaptive systems based on decision-theoretic planning It is not clear if the benefit of applying decision-theoretic planning in a user-adaptive system is actually appreciated by the users. Empirical studies are necessary to get feedback about how the users react to user-adaptive systems based on decision-theoretic planning. Such empirical studies will address questions like:

- *In what kind of realistic scenario can a user-adaptive system be implemented and tested?*

  In order to study how users can cope with a system based on the decision-theoretic planning approach, we need to build a complete prototype of the system for a realistic scenario. We have identified a PDA-based location-aware shopping guide as one interesting example application. The system should help its user to buy the items on his shopping list more efficiently by taking expected product availability and expected waiting times at cash desks into account.

- *How do users get along with a system giving recommendations based on decision-theoretic planning and how is the acceptance of such a system?*

  Once we have implemented a complete prototypical system, for example, for a shopping scenario, we can conduct user studies to find out about the usability and the user acceptance of the corresponding system. The feedback about the users' experience with the system can be collected in questionnaires and the user's performance with the corresponding system can be evaluated in comparison with alternative approaches for the corresponding task.

- *How can recommendations based on decision-theoretic planning be explained and justified?*

  It may be difficult for users to comprehend recommendations based on decision-theoretic planning. The decision-theoretic planning process itself offers nothing but the notion of highest expected utility of a recommendation to justify a decision. The information content of a justification based on highest expected utility will in general not be satisfying for a user. Therefore, we need to develop appropriate methods to generate more descriptive explanations of decision-theoretically planned recommendations.

Special requirements of specific types of application Some example applications that we want to consider in this thesis include special problems whose solutions may not be relevant for user-adaptive systems in general. The special requirements of such specific applications bring up the following interesting research questions:

- *What is the relationship between decision-theoretic planning and user modeling?*

  Many user-adaptive systems maintain a user model that may comprise information relevant for a decision-theoretic planning process. For example, the planning of the right form of output presentation for a sequence of recommendations may depend on parameters like the user's cognitive load or his subjectively felt time pressure. We will explore how such information can influence a decision-theoretic planning process in the context of a user-adaptive system. As a user's cognitive load and subjectively felt time pressure are not observable directly, this line of research may bring up a way to consider partially observable information in the complex decision-theoretic planning for a user-adaptive system without having to consider partial observability in the model for the decision-theoretic planning explicitly.

- *How can the planning of navigation recommendations be integrated with the planning of their presentation?*

  One type of interaction that we are especially interested in is concerned with location-aware adaptive navigation recommendations. The planning of navigation recommendations and the planning of the presentation of navigation recommendations are typically not independent from each other. We will analyze a scenario for which dealing with the navigation planning and the presentation planning in separate planning processes may yield suboptimal results. To avoid suboptimal results, we develop a decision-theoretic model that allows an integrated planning of navigation recommendations and their presentation.

- *How can decision-theoretic planning of navigation recommendations and their presentation be combined with traditional route planning?*

  Where navigation recommendations are involved, it is possible that a preferred route exists, for example, a route that the user is already familiar with or a route that has some other special properties (e.g., a minimized number of turning points). We show how the presentation of predefined routes can be augmented by a decision-theoretic planning approach.

1.6 Outline

Chapter 2 surveys different approaches to decision making in user-adaptive systems. We analyze systems that make only individual decisions, and describe general approaches for individual decision making. We give an overview of different planning approaches and describe how far these approaches have already been applied in user-adaptive systems. **Chapter 3** gives an introduction to decision-theoretic planning and illustrates the main concepts with a simple example from the context of user-adaptive systems. We describe the elements of Markov decision processes and put some emphasis on different state space representations. Moreover, general complexity issues will be considered in the third chapter. **Chapter 4** explains the modeling of the fully observable Markov decision processes that we use in both a stationary and a mobile scenario of low complexity. In the stationary scenario, we describe a system that gives the user instructions about how to operate a technical device. The mobile scenario is based on an airport assistance system that guides a passenger to the gate and thereby tries to take a secondary goal of the passenger into consideration. We show how both models can be coupled with a user modeling component. In the context of airport navigation assistance, we explain how decision-theoretic planning and other approaches for navigation planning can be combined. Moreover, we describe a method that guarantees that the interaction meets a fixed deadline. **Chapter 5** addresses the scalability of the decision-theoretic approach. We describe a hierarchical decision-theoretic planning approach based on goal priorization to cope with complex planning problems. In the context of airport navigation assistance, we demonstrate goal-priorized decision-theoretic planning involving two planning layers. In this example, we apply a macro decision-theoretic planning process to break down the complex planning problem into several tractable decision-theoretic planning problems on the micro level. Eventually, we show how to deal with fixed deadlines in hierarchical decision-theoretic planning. **Chapter 6** presents the workflow for the development and application of decision-theoretic planning in the context of user-adaptive systems. We describe the workflow on the basis of examples of low complexity and eventually extend the workflow description such that it is applicable to more complex problems. **Chapter 7** focuses on two usability issues. The first issue is user acceptance: We describe the development of a decision-theoretic shopping guide on a commercially available hand-held computer and the experience that subjects made with the system in two user studies. The second issue is explainability of decision-theoretically planned navigation recommendations. We report the results of a study in which we discovered what kind of recommendations users wish, and we examine how appropriate explanations can be generated. **Chapter 8** concludes with a summary of contributions of this thesis. We review the main limitations of the decision-theoretic planning approach in the context of user-adaptive systems and point out to some interesting open questions.

2 Decision Making and Planning in User-Adaptive Systems

There exist various approaches to the problem of making decisions in user-adaptive systems. In this chapter, we will give on overview of the most prominent alternatives to decision-theoretic planning, which have been applied in user-adaptive systems so far. First, we consider systems that do not plan ahead but rather make individual decisions about actions to be performed. We discuss three approaches to making individual decisions and thereby focus on the decision-theoretic methods.

The potential of individual decision making is limited. There are undoubtfully situations in which the performance of a system can be improved by employing planning to look several steps ahead. This way, a system can take the future consequences of a decision into consideration. We give a selective overview of the spectrum of planning approaches that have been applied in user-adaptive systems already, and we expose the potential of decision-theoretic planning in this context.

Some systems that we describe combine several techniques and could therefore be addressed in more than one of the following sections. We assigned them to the individual sections according to the aspects that we want to emphasize about the system. For example, the PRACMA system is described in a section about systems that rate alternatives, but it would also have fit to the section on decision-theoretic methods for individual decisions.

2.1 Systems That Make Only Individual Decisions

Among the approaches most frequently used for decision making in intelligent user interfaces, Jameson et al. (2001) distinguish three principle types. In the following, we use the same categorization to discuss our selection of user-adapted systems. The three types can be summarized as (1) observing and imitating the user, (2) rating alternatives in terms of their relevance with respect to the user's needs, and (3) using decision-theoretic methods to evaluate the consequences of decisions. As the examples will show, each of the three types of individual decision making can be implemented in very different ways. We go into some more detail with regard to the decision-theoretic methods, as they represent the core of research in the READY project and provide the background for a better understanding of the decision-theoretic planning methods.

2.1.1 Systems That Imitate the User

It is a very attractive approach to have user-adaptive systems learn from the user himself. There are a lot of scenarios in which the user of a system has to perform the same or similar sequences of actions over and over again. A user-adaptive system that can detect such recurrent sequences and perform the corresponding tasks automatically is appealing. A very natural approach to achieve such automation is to have a system learn from the user's behavior like a human apprentice would do—by "looking the user over the shoulder". The main job done by an *apprenticeship model* is the *prediction* of the action that the user would most likely be doing. Two examples for this approach are Maxims, an email assistant, and Eager, a system that detects and anticipates repetitive tasks in the HyperCard environment on Apple Macintosh computers.

2.1.1.1 Maes (1994): *Maxims*, an Electronic Mail Assistant

Maxims (Maes, 1994) is a system that assists the user with electronic mail. The Maxims agent learns to prioritize, delete, forward, sort and archive email messages on behalf of the user. The agent continuously "looks the user over the shoulder" while the user deals with email. It memorizes the situation-action pairs generated while the user works with the system. For example, if the user saves a particular email message after having read it, the email agent adds a description of the situation and the action taken by the user to its memory of examples.

When a new message arrives, the agent tries to predict the action(s) of the user based on the examples stored in its memory. The agent compares the new situation with the memorized situations and tries to find a set of *nearest neighbors* (or close matches). The most similar of these memorized situations contribute to the decision of which action to take or suggest in the current situation. The distance metric used is a weighted sum of the differences for the features that make up a situation. Some features carry more weight than others. The weight of a feature is determined by the agent: Occasionally (e.g., at night), the agent analyzes its memory and determines the correlations between features and actions taken. For example, the agent may detect that the "from" field of an email message is highly correlated to whether its user reads the message, while the "date" field is not correlated. The detected correlations are employed as weights in the distance metric.

Maxims has some properties in common with today's spam filters. However, while Maxims learns from the user to predict the correct folder for all types of incoming emails, spam filters are specialized in detecting spam emails. To improve their performance, spam filters can learn from what the user recognizes as spam, but they typically employ a multitude of predefined filters already right from the start.

The Maxims agent does not only predict which action is appropriate for the current situation, but it also measures its confidence in each prediction. The confidence level is determined by (1) whether or not all the nearest neighbors recommended the same action, (2) how close or distant the nearest neighbors are, and (3) how many (similar) examples the agent has memorized (a measure of the accuracy of the correlation weights). Two thresholds determine how the agent uses its predictions. When the confidence level is above the "do-it" threshold, then the agent autonomously takes the action on behalf of the user. In that case, it writes a report for the user about the action it automated. The user can ask the agent for its report of automated actions at any time. If the confidence level is above the "tell-me" threshold, then the agent will offer its suggestion to the user but will wait for the user's confirmation to automate the action. The user is responsible for setting the ”tell-me” and ”do-it” thresholds for actions at levels the user feels comfortable with. For example, if the user feels paranoid about the agent autonomously deleting messages, then the user can set the ”do-it” threshold for that action at a maximum.

The Maxims agent gradually gains competence by observing the user and acquiring more examples. As there are no or just a few examples available at the beginning, there are two additional competence acquisition schemes: (1) It is possible for the user to instruct the agent explictly, and (2) the agent can asks for help from other agents that are assisting other users with electronic mail.

2.1.1.2 Cypher (1991): *Eager,* Programming Repetitive Tasks by Example

Cypher (1991) describes the *programming by example* system Eager for the HyperCard environment on Apple Macintosh computers. Eager constantly monitors the user's interaction with the HyperCard environment and tries to complete the interaction automatically when it detects an iterative pattern.

Eager highlights menus and objects on the screen to indicate what it expects the user to do next. As users continue to perform their activity, they will notice that the objects they are about to select have already been highlighted by the system. When it becomes apparent that Eager knows how to perform the task correctly, they can tell Eager to complete the task for them, that is, when Eager is correct, the user's intended selection will appear highlighted in green before it is selected. When Eager is incorrect, the user's choice will not match the highlighted item, and Eager will use the actual choice to revise its program.

The use of *anticipation* allows Eager to interfere minimally with the user's normal activities. Eager is written so that it never asks the user for information; the only information comes from recording the user's ongoing actions. In particular, the user does not signal the start of an example, and never explicitly confirms or rejects a hypothesis. Performing an action that matches the anticipation is implicitly confirming, and performing any other action is implicitly rejecting Eager's anticipation.

Eager is written in LISP and runs as a background application. It receives information about high-level user events in the HyperCard environment via interprocess communication. Whenever a new event is reported, Eager searches through previous events in the history to find one that is similar.

When two similar events are found, Eager assumes that the second event marks the beginning of the second iteration in the loop. All preceding events, back to the first similar event, are presumed to constitute the first iteration in the loop. Eager now monitors each new incoming event to see if it is similar to the corresponding event from the first iteration. If patterns can be found for each pair of events, Eager concludes that it has detected an iterative loop, and the Eager icon pops upon the screen. Based on the generalizations formed from these two iterations, Eager instantiates the next steps in the pattern and directs HyperCard to highlight the appropriate items in green.

2.1.1.3 Applicability and Limitations

A requirement for the applicability of the methods described in Section 2.1.1 is that the user knows very well how he should/wants to act in a given situation. The method can be used in situations in which the system's assistance consists mainly in automating (more or less) simple tasks, that otherwise would not put the user in front of a challenging problem, but would rather annoy the user because of their simplicity and monotony. The method is not based on an explicit model of the user and does not make inferences about the user's current situation nor about the current state of the interaction of the system with the user. Future consequences of the system's decision are not taken into account, for example, what will happen if Maxims automatically deletes or forwards an email incorrectly one, or two, or three times? However, the method is well suited, for example, to provide some sort of guidance or to enable a system to automate and take over some tasks from the user, thereby allowing him to work with a system more conveniently. Moreover, such systems have the potential to improve over time: the more examples are available to the system, the better will its performance be.

2.1.2 Systems That Rate Alternatives

Systems that recommend documents, products, or other objects to the user often do that on the basis of certain features of the recommended entity. A frequent approach is to rate a large number of entities, for example products, in terms of their relevance with respect to the user's desires—as expressed, for example, in the user's query. Sometimes a decision is not based on a single rating, but on a combination (e.g., a weighted sum) of ratings. *Multi-attribute utility theory* provides a mathematical framework for this approach.

2.1.2.1 Multi-Attribute Utility Theory

The basic idea of the *multi-attribute utility theory* (von Winterfeldt & Edwards, 1986) is simple. It reflects a common human decision making strategy for the selection of one option out of many. The value function $v(x)$ of an object (e.g., a product that one might want to buy) or a characteristics (e.g., of a hotel that one might want to book) x is defined as a weighted addition of its evaluation with respect to its relevant *value dimensions*:

$$v(x) = \sum_{i=1}^{n} w_i v_i(x). \tag{2.1}$$

For example, suppose one would like to go on holiday and to select an appropriate hotel. There are typically a number of relevant aspects that influence the decision on what hotel to choose. For example, for the evaluation of a hotel, one can consider: How expensive is the hotel? How close is it to the beach? What sports facilities does it offer? How comfortable is the hotel? In equation 2.1, w_i is the relative importance of the dimension d_i, and $v_i(x)$ is the evaluation of x on the i-th value dimension d_i, defined as

$$v_i(x) = \sum_{a \in A_i} w_a v_a(l(a)), \tag{2.2}$$

where A_i is the set of all attributes relevant for dimension d_i, w_a is the relative importance of attribute a for dimension d_i, and $v_a(l(a))$ is the evaluation of the actual level[1] $l(a)$ of attribute a on dimension d_i. For example, for the evaluation of the comfort a hotel offers, one can consider: What is the size of the rooms? Are the rooms air conditioned? Do they have a balcony?

In the following, we survey three systems that rate alternatives: PRACMA, an interactive electronic sales agent that uses MAUT to adaptively advertise a car; Letizia, an interface agent that recommends web pages; and Apt Decision, an assistant for rental real estate decisions that learns the user's preferences from his interaction with the system.

[1]In order to evaluate attributes, it is necessary to construct a scale representing the properties of the *levels* of an attribute, for example, a scale from 0 (very small) to 10 (very large) specifying the room size.

2.1.2.2 Jameson et al. (1995): PRACMA, a Car Purchasing Dialog System

PRACMA[2] (Jameson, Sch¨afer, Simons, & Weis, 1995) is a dialog system for purchasing a car. PRACMA builds a profile about the customer's preferences and acts as a consultant. The user can specify his requirements for a car in a language that is familiar to him, that is, he does not have to know about any technical terms. The system identifies the user's desires and infers his preferences about product categories or features the desired car should offer.

PRACMA uses *multi-attribute utility theory (MAUT)* to represent each customer as a vector of weights on multiple evaluation dimensions. The total value of an object is the sum of the weighted evaluations on each dimension. Dimensions of evaluation include, for example, how ecological or how sporty a car is. The PRACMA system assigns these attributes to the different types and models of cars of the brand and estimates the importance of an attribute for a specific dimension. The customer's preferences for certain dimensions are inferred from observations about evaluations of attributes with *Bayesian networks* (see Section 2.1.3.1). For example, from a negative assessment of the lack of air conditioning, the system draws the inference that comfort is a rather important dimension for the current customer.

2.1.2.3 Lieberman (1995): *Letizia*, an Assistant for the Web

Letizia (Lieberman, 1995) is a user interface agent that assists a user browsing the Web. It was developed at about the same time as WebWatcher (see Section 2.2.6.3) and provides a very similar service. It tracks the behavior of its user and attempts to anticipate items of interest by doing concurrent, autonomous exploration of hyperlinks from the web page which the user currently visits. Unlike WebWatcher, Letizia does not ask the user about his interests at the beginning of a session but tries to infer them from the user's browsing behavior. Letizia conducts a resource-limited search through the space of web pages in parallel with the user's browsing. This way, the user may request a set of recommendations at any time. While a user of WebWatcher gets recommendations automatically as soon as there are recommendable links available on a page, the Letizia user has to ask for recommendations explicitly if he is unsure about what to do next.

Letizia does not use information about the browsing behavior of other users. The inferences about appropriate links to recommend are drawn using heuristics about the meaning of certain browsing behaviors of the current user only. For example, saving a reference to a document indicates interest in the topic of that document, whereas passing over a link indicates that the topic of that link is rather uninteresting for the user (within the current session). Letizia's content model of a document is simply a list of keywords. Rather than to estimate the importance of a document for a user (e.g., by assigning an importance probability to it), Letizia determines a preference order of interest among a set of links. Letizia can explain why it has chosen a document in that it refers to anticipated topics of interests that correspond to the recommendation.

2.1.2.4 Shearin & Lieberman (2001): *Apt Decision*, Rental Real Estate Decisions

The Apt Decision agent (Shearin & Lieberman, 2001) learns user preferences in the domain of rental real estate by observing the user's critique of apartment features. The agent uses interactive learning techniques to build a profile of user preferences. In the beginning, the system presents a list of sample matching apartments using a sparse initial profile provided by the user. Thereby,

[2]PRACMA stands for **PR**ocessing **A**rguments between **C**ontroversially-**M**inded **A**gents.

the user can discover a new feature of interest and assign this feature a new weight. The user can emphasize the importance of a particular feature by referring to it repeatedly. New profile information can also be derived automatically when the user expresses his preference with regard to one of two sample apartments proposed by the Apt Decision agent on request.

When the user is finished examining the sample apartments, he has a profile of apartment preferences that can be saved to a file. The profile can be used to retrieve matching apartments from the set provided with the agent, or it can be taken to a human real estate agent as a starting point for a real-world apartment search. As the user modifies the profile, the system updates the weights on its representation of the ideal apartment and re-orders the potential matches in the data set to reflect the new weighting.

2.1.2.5 Applicability and Limitations

The methods described in Section 2.1.2 are well suited for situations in which the decision about what option (possibly out of many options) to choose is influenced by several (and possibly many) factors. The impact of the relevant factors can vary, and the methods explicitly (e.g., Letizia) or implicitly (e.g., PRACMA) determine a preference order among the available options. In the examples, we have seen how a user is modeled in terms of his interests. Similar to the methods described in Section 2.1.1, also the methods described in this section do not take future consequences of actions into account. For example, Letizia, although looking ahead in that it carries out a search in Web space, cannot take into account what the consequences of recommending a “bad” hyperlink will be for the further course of interaction with the user. Similarly, PRACMA is only able to reason about what the consequences of an inappropriate question or recommendation might be once it applies the method of *global anticipation feedback*, which we will describe in Section 2.2.5. But even then, PRACMA’s look-ahead is limited to a small number of future steps.

2.1.3 Systems That Use Decision-Theoretic Methods

Decision theory (see, e.g., Russell & Norvig, 2003, Part V, for an introduction) is a branch of statistics concerning strategies for decision making in non-deterministic systems. Decision theory combines *probability theory* and *utility theory* in that it seeks to find strategies (based on uncertain actions) that maximize the expected value of a utility function measuring the desirability of possible outcomes. The fundamental idea of decision theory is that *an agent is rational if and only if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action it chooses (principle of maximal expected utility, MEU)*. Decision-theoretic methods are most applicable to problems in which a system (a) has a small number of possible actions to achieve a given result, and (b) the system’s decision whether to perform an action or not is a yes-or-no decision (Jameson et al., 2001). In particular, such problems tend to be fine-grained and each decision represents only a small part of the system’s overall processing, while the overall impact of many such decisions can be important. We give several examples for the decision-theoretic approach that come from the Microsoft research labs: the Lumière project, which sets the stage for the Microsoft Office Assistant, Microsoft’s Presenter for the Powerpoint application, and Microsoft’s Bayesian Receptionist. Moreover, we will describe the decision-theoretic tutoring system *DT Tutor*. The systems reviewed in this section are similar to the READY prototype in that they all use Bayesian networks or dynamic Bayesian networks to make inferences about the user’s situation or the state of the interaction of the system with the user.

2.1.3.1 Bayesian Networks and Influence Diagrams

Pearl (1988) introduces *Bayesian networks (BNs)*[3] as an inference mechanism to draw conclusions from uncertain knowledge. The main mathematical equation underlying the inferences with Bayesian networks is known as *Bayes' rule*[4] and describes how to compute the conditional probability of a proposition B given evidence A:

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}. \tag{2.3}$$

$P(A \mid B)$ reads as "the probability of A given that all we know is B". What makes Bayes' rule very useful in practice is that there are many cases where good probability estimates for $P(A), P(B)$, and $P(A \mid B)$ are available to compute $P(B \mid A)$. This advantage makes Bayesian networks superior to working on joint probability distributions of random variables—an approach which was used before the advantages of using conditional probabilities were recognized. If one describes the conditional probability of a proposition B given evidence A by

$$P(B \mid A) = \frac{P(B \wedge A)}{P(A)}, \tag{2.4}$$

then one has to maintain a possibly huge joint probability table to compute $P(B \mid A)$. The joint probability distribution over n Boolean variables is a table with 2^n entries.

A Bayesian network typically represents uncertain dependencies between random variables, for example in READY, symptoms and possible causes for these symptoms. A Bayesian network consists of a *directed acyclic graph*, whose nodes represent random variables, and a set of *conditional probability tables*, each of which is associated with one of the nodes in the graph. Each of the directed links connects exactly one pair of nodes X and Y and can be interpreted as "X has a direct influence on Y". The conditional probability table of a node quantifies the influence that its parents have on the node.

Figure 2.1 shows a simple Bayesian network that was used in an experimental setting within the READY project—the experiment is described in detail in Section 3.1.1. A subject is supposed to follow some simple instructions given by a computer system. The system can give instructions either stepwise or in bundles of two, three, or four instructions. Sometimes the subject is distracted while executing the instructions. The Bayesian network of Figure 2.1 can be used to make inferences in two directions: (1) to make predictions about the effect of certain circumstances under which the subject executes instructions; for example, knowing that the instructions are given in bundles of two and that the subject feels distracted with a certain probability, we can use the BN to predict how much time the subject will need to execute the instructions and what the probability is that the subject makes an error; (2) to reason about the causes for observed symptoms; for example, knowing that the subject needed a certain amount of time to execute some instructions and made an error, we can use the BN to estimate the probability that the subject was given the instruction stepwise or bundled and if the subject was distracted while executing the instructions. It is also possible to instantiate both cause and symptom variables. For example, knowing that the subject was instructed stepwise, that he needed a certain amount of time to execute the instructions, and that he made an error, we can estimate the probability that the subject was distracted.

[3]Bayesian networks are also called *belief network, probabilistic network,* or *causal network.*

[4]The Bayes' rule is also known as *Bayes' law* or *Bayes' theorem.*

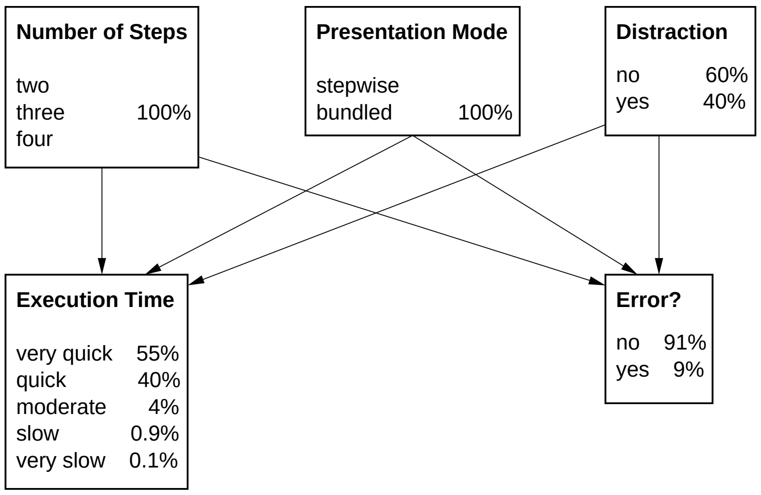


***Figure 2.1**: A simple Bayesian network for an experimental setting.*
(The BN represents causal dependencies between symptoms—the variables 'Execution Time' and 'Error?'—and causes—the variables 'Number of Steps', 'Presentation Mode', and 'Distraction'. The histogram for each variable shows a probability distribution that represents the system's belief about the value of that variable.)

The Bayesian network of Figure 2.1 can be extended to an influence diagram (see, e.g., Howard & Matheson, 1984 or Neapolitan, 1990) and thereby be used by a system to make decisions. Two steps are necessary: (1) A *utility node* has to be added, which expresses the systems evaluation of a particular combination of values of the dependent variables 'Error?' and 'Execution Time', and (2) the node 'Presentation Mode' has to be turned into a *decision node*. Step two makes explicit that the system can freely choose the value of this variable instead of just forming a belief about it. The system can read out the optimal decision about the presentation mode directly from the corresponding decision node. The values of the two alternatives (in principle there can be more than two options) specified in the decision node are determined by evaluating the complete influence diagram on the basis of the utility declared in the utility node. The resulting influence diagram is shown in Figure 2.2.

Influence diagrams are also used in the context of two or more decisions in sequence. In this case, a decision node is added to the influence diagram for each decision. The determination of a decision influences later decisions—the influence of future decisions on the current decision cannot be considered.

2.1.3.2 Horvitz et al. (1998): *Lumière* and *Microsoft Office Assistant*

Research in the Lumière project provided the basis for the Microsoft Office Assistant in the Microsoft Office '97 environment (Horvitz, Breese, Heckerman, Hovel, & Rommelse, 1998). The assistant's overall goal is to take automated actions in office applications and thereby optimize the user's expected utility of the interaction with these applications. For example, the assistant can detect if a user applies inefficient command sequences and call his attention to a more effective alternate sequence or a shortcut for the task. A system taking autonomous actions to assist users needs to balance the benefits and the costs of such actions. Poor advice can be quite costly to users:

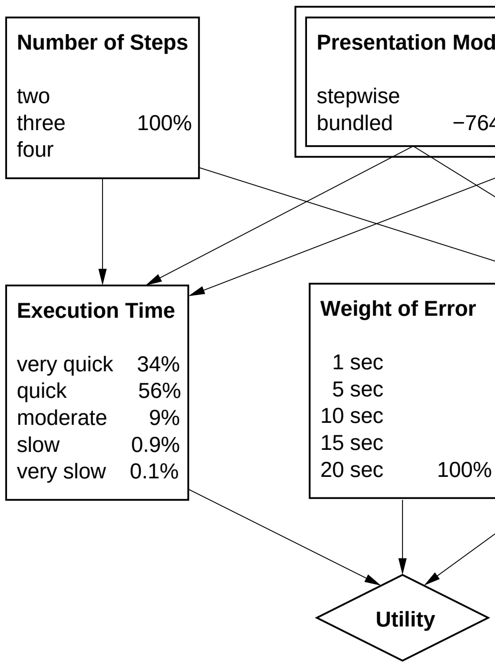


Figure 2.2: *Extending the Bayesian network of Figure 2.1 to a simple influence diagram.*
(A utility node has been added, and 'Presentation Mode' has been turned into a decision node. The expected utility of the bundled presentation of three instructions given DISTRACTION? is -7647. Apart from the minus sign, the value is essentially the sum of (a) the expected execution time (in msec) and (b) a penalty that reflects the possibility that the user will make an error. By comparing this utility with that of stepwise presentation, the system can decide which mode to use. The variable 'Weight of Error' represents the amount of additional execution time that would be worth accepting in order to avoid an error.)

They can become distracted by the advice and get annoyed about the system's performance.

The project explores Bayesian user models to infer a user's needs from information about his background, actions, and queries. Using Bayesian networks and influence diagrams, the assistant can take actions based on probability distributions over the user's goals.

2.1.3.3 Paek & Horvitz (2000): *MS Presenter* and *MS Bayesian Receptionist*

Based on the research in the Lumière project, Paek and Horvitz (2000) propose the task independent architecture *Quartet* for supporting robust continuous spoken dialog. The architecture was used for Presenter, a prototype system for navigating Microsoft Powerpoint, and the Bayesian Receptionist, a prototype system for dealing with tasks typically handled by front desk receptionists at the Microsoft corporate campus.

The idea behind the *Quartet* architecture is to put much emphasis on the *grounding* of a dialog. Grounding is the process by which dialog partners try to coordinate the presentation of their utterances to establish, maintain, and confirm mutual understanding (Clark & Brennan, 1991). In *Quartet*, the process of grounding is treated as decision making under uncertainty. Again, key uncertainties are represented with Bayesian networks. Local expected utility and value-of-information analyses are used to identify actions that maximize mutual understanding. *Quartet* supports grounding on different levels of understanding. The selected grounding strategy is refined using value-of-information analysis to ask questions, make recommendations, and seek out information in a dialog setting. The system calculates for every observation the expected utility of the best decision associated with each value the observation can take considering the likelihood of different values. Once value-of-information analysis recommends a piece of evidence to observe, the system can explicitly provide that recommendation or phrase a question to solicit that information.

2.1.3.4 Dynamic Bayesian Networks and Dynamic Decision Networks

Dynamic Bayesian networks (DBNs)[5] extend Bayesian networks in that they allow to consider the changes of random variables over time. Dynamic Bayesian networks have a generic structure: They are composed of time slices (each of which contains a static Bayesian network) for each discrete time step up to the current time step t. There are links that connect nodes of consecutive time slices. Dynamic Bayesian networks are used to calculate the probability distribution for the state at time t, considering all evidence that was collected in earlier time slices up to time t.

In analogy to the extention of Bayesian networks to influence diagrams, dynamic Bayesian networks can be extended to *dynamic decision networks (DDNs)* by adding utility nodes and decision nodes for actions. While dynamic Bayesian networks are used (e.g., also in the READY prototype) to interpret events that occurred in time steps up to time step t, dynamic decision networks allow to calculate a decision for time t that maximizes an agent's expected utility over the remaining state sequence. Thus dynamic decision networks provide approximate solutions for partially observable Markov decision processes (cf. Section 3.1.6). The degree of approximation depends on the amount of look ahead. While dynamic decision networks cannot be used directly to compute a policy (a mapping from states or belief states to actions, cf. Section 3.1.4), they can be used as a means of representation for the dynamics of a partially observable process. Russell & Norvig, 2003, Part V, Chapters 15.5 and 17.5, give a much more detailed introduction to dynamic Bayesian networks and dynamic decision networks.

2.1.3.5 Murray et al. (2001): *DT Tutor*, Selecting Tutorial Discourse Actions

Murray, Van Lehn, and Mostow (2001) describe a decision-theoretic approach for selecting tutorial discourse actions. *DT Tutor* uses a dynamic decision network to consider the tutor's objectives and uncertain beliefs in adapting to and managing the changing tutorial state. The system tries to predict both the tutor's and the student's next actions and their effects on the tutorial state, that is, the discourse between the tutor and the student and their progress at completing the tutorial tasks. The tutor might have various objectives at the same time, for example, increasing the student's knowledge, helping the student achieve a task, or reassure the student's affective state.

DT Tutor represents the tutor's uncertain beliefs by means of Bayesian networks, the changes of the tutorial state over time by means of dynamic Bayesian network (DBN), respectively. Extending the DBN with decision nodes representing the tutorial action alternatives and utility nodes representing the tutor's preferences among the possible outcomes results in a dynamic decision network, which is used to determine what tutorial action to take next in order to achieve the highest expected utility. In this respect *DT Tutor* and PRACMA (cf. Section 2.1.2.2) are very similar.

[5]DBNs are also called dynamic belief networks.

2.1.3.6 Applicability and Limitations

The decision-theoretic methods described in this section typically represent a part of a system's user modeling component, which can be extended to compute the optimal next action to take. The methods are applicable in situations where probabilities and utilities of possible outcomes of actions can be quantified—the more accurate the numbers are, the more reliable are the system's predictions and decisions. If accurate numbers are available, the system—assuming a correct modeling of the dependencies in the problem at hand—can act in the way that yields the maximum utility for the user. The identification of the dependencies is a non-trivial problem in many domains. Wittig (2003) describes an approach for the learning of Bayesian networks to overcome these difficulties in the context of user-adaptive systems.

One general advantage of decision-theoretic methods for decision making is that they fit in well with decision-theoretic methods for inference under uncertainty, such as Bayesian networks. Section 2.1.3.1 explained how a Bayesian network can easily be extended to an influence diagram by adding utility and decision nodes.

The dynamic methods (DBNs and DDNs) can be used to track the state of the user or the state of the interactions between the system and the user over time. This characteristic allows to consider the complete history of an interaction for the decision about the next step. As the *DT Tutor* system illustrates, dynamic decision network can be used to anticipate the student's next action in order to decide on the systems next action. DDNs could in principle be used to plan several steps under partial observability ahead (as already mentioned in Section 2.1.3.4). However, known algorithms that work on DDNs are not effective for this purpose (cf. also Russell & Norvig, 2003, Part V, Chapter 17.5).

2.1.3.7 Other Approaches to Reasoning with Uncertainty

AI has considered other approaches to deal with uncertainty (see, e.g., Russell & Norvig, 2003, Part V, Chapter 14.7, for a selection). Among these, Dempster-Shafer theory and fuzzy logic belong to the most successful ones. *Dempster-Shafer theory* (Dempster, 1968; Shafer, 1976) considers the degree of belief that an agent has about the probability of a proposition. Consider the following example: For flipping a fair coin, we regard the probability of 0.5 as a reasonable probability for heads. For flipping a coin that is biased, but we do not know which way, probability theory would still propose a probability of 0.5 for heads. Dempster-Shafer theory distinguishes these two cases and thereby makes a difference between uncertainty and ignorance. Rather than computing the probability of a proposition, it computes the probability that some evidence supports a proposition. In the context of user-adaptive systems, Dempster-Shafer-theory has been applied, for example, for the assessment of UNIX expertise (Petrushin & Sinitsa, 1993) and for the recognition of an email user's plans (Bauer, 1996).

Decision making with Dempster-Shafer theory often requires initially some ad hoc assumptions or decisions on the part of the designer. As indicated by Bauer, 1996 (Section 8) and summarized in comparison with Bayesian networks by Jameson, 1996 (Section 3.1.1.), relying on weak assumptions for the plan recognition process in the beginning complicates decision making after the plan recognition process. Using a Bayesian network, each hypothesis is associated with a single probability, while in Dempster Shafer theory, it happens that there are several different measures of the extent to which a set of hypotheses is compatible with the evidence.

Fuzzy logic (Zadeh, 1965, 1968) introduces a notion of *vagueness*. Instead of claiming that events are either true or false—like both first-order logic and probability theory do—fuzzy logic allows events to be “sort of” true. That is, while probability theory is concerned with degree of belief[6], fuzzy logic works on degrees of possibility. Fuzzy logic is based on *fuzzy set theory*, which implements classes of data with boundaries that are not sharply defined. A fuzzy subset A of a set X is characterized by assigning to each element x of X the degree of membership of x in A (e.g., X is a group of people, A the fuzzy set of old people in X). If X is a set of propositions, then its elements may be assigned their degree of truth, which may be ’absolutely true’, ’absolutely false’, or some intermediate truth degree: Some proposition may be more true than some other, for example, in the case of a vague (imprecise) proposition like ’this person is old’. In the context of user-adaptive systems, fuzzy logic has been applied, for example, in the user modeling component *Knome* of the *Unix Consultant* system (see, e.g., Chin, 1989), and in the *Sales Assistant* (Popp & L¨odel, 1996) to predict how a user will evaluate a particular product.

Popp and L¨odel (1996) illustrate how a sort of multi-attribute utility theory can be realized with fuzzy logic. The fuzzy membership functions that they use are similar to value functions within MAUT. Even if they do not explicitly arrive at decisions with these rules, one could do so by treating the recommendations to the user as decisions made by the system.

We refer to Jameson (1996) for a broader survey of user modeling techniques. The survey includes systems that use Dempster-Shafer theory or fuzzy logic, discusses several aspects of the usability of such techniques for user modeling, and compares the two techniques to Bayesian networks.

2.2 Planning in User-Adaptive Systems

The systems described in Section 2.1 have a great deal to offer in situations where it is sufficient to look a single step ahead. Some applications do not necessarily require a look ahead, for example, email assistance as provided by Maxims or recommendations for rental real estates as provided by Apt Decision. Other applications can undoubtedly benefit from looking ahead, as the following section will show. In such applications, the quality of the system behavior can typically not be determined locally (in terms of the quality of the next decision to take) but has to be optimized globally (in terms of the result of a whole sequence of decisions). The optimization of a sequence of decisions becomes particularly complex if the results of the actions taken cannot be predicted with certainty. This uncertainty almost always evolves if a system tries to take the user’s actions into account. In this section we will survey both systems that assume complete determinism in the results of actions and systems that face up to the uncertainty in the results of actions taken by the user. Moreover, we consider systems that—at runtime—assume perfect knowledge about the state of the interaction and eventually systems that try to deal with partial observability of the interaction state.

[6]Probability theory is the mathematical study of probability. Two crucial concepts in the theory of probability are the random variable and the probability distribution of a random variable. Some statisticians will assign probabilities only to events that they think of as random, according to their relative frequencies of occurrence, or to subsets of populations as proportions of the whole; those are frequentists. Others assign probabilities to propositions that are uncertain according either to subjective degrees of belief in their truth, or to logically justifi able degrees of belief in their truth. Such persons are Bayesians. A Bayesian may assign a probability to the proposition that there was life on Mars a billion years ago, since that is uncertain; a frequentist would not assign such a probability, since it is not a random event that has a long-run relative frequency of occurrence. From Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Probability_theory`.

2.2.1 Plan Libraries

The interaction of a system (e.g., an assistant or a tutor) with different users often follows the same patterns over and over again. These patterns can be represented as raw plans and collected in libraries. Such off-the-shelf plans (sometimes called scripts, recipes, or the like) specify the raw structure of an interaction and usually have to be instantiated for a particular application or a particular session. We review the use of plan libraries in this section because in the broadest sense the selection of a plan from a library (i.e., in particular the reuse and possibly the adaptation of a previously computed plan) can be considered as an effective approach to planning for some domains. The applicability of plan libraries is very limited because of the inflexibility and insufficient scalability of the approach. Plan scripts were also used in READY (in particular during the first and second phase of the collaborative research program). However, scripts became less important in READY as many planning problems in the context of READY are meanwhile solved with decision-theoretic planning. We describe two approaches that use (not exclusively, though) plan libraries: (1) The use of plan libraries in the PHI system illustrates how the effort for repeatedly generating similar plans can be reduced by modifying already existing plans; (2) the COLLAGEN architecture is the general basis for a collection of applications that use recipes to interpret the user's actions and react to them.

2.2.1.1 Bauer et al. (1993): PHI, a Logic-Based Tool for Intelligent Help Systems

Bauer, Biundo, Dengler, Koehler, and Paul (1993) combine plan generation and plan recognition in their logic based tool PHI to supply help systems with services like *semantic plan completion*. They illustrate the tool with examples taken from the UNIX mail domain. The plan generation component can either plan from first principles or reuse already existing plans (i.e., do planning from second principles—see K¨ohler, 1994, and Koehler, 1996). The plans provided serve as plan hypothesis in the plan recognition process. While observing the user's actions step by step, the plan recognizer tries to confirm the plan hypotheses by proving that the action sequence observed up to the time being is an admissible instance of the current plan hypothesis.

PHI is based on the *logical language for planning* (LLP). The plan generation is done deductively (using a sequent calculus for LLP) and the plan recognition is done abductively.

The reuse of the plans is a particularly interesting aspect of the PHI system. The planner tries to reuse plans to avoid the repetition of previous planning effort. It thereby searches for a plan in the library, which can be reused for the planning problem at hand with some modifications. The plan is repeatedly modified until the formal proof of the current plan succeeds (i.e., at least the preconditions required by the plan hold in the current situation and at most the goals achieved by the plan are required as current goals). However, the effort for modifying an existing plan can in some cases be enormous.

2.2.1.2 Rich & Sidner (1998): COLLAGEN, a Collaborative Agent Middleware

COLLAGEN[7] (Rich & Sidner, 1998; Rich, Sidner, & Lesh, 2001) is a middleware for building interface agents based on collaborative discourse theory (Lochbaum, 1998). COLLAGEN is used as the basis for intelligent tutoring systems (such as *Paco*, a simulation-based training system for operating gas turbine engines that propel naval ships described by Rickel, Lesh, Rich, Sidner, &

[7]COLLAGEN stands for **COLL**aborative **AGEN**t.

Gertner, 2002) and assistance systems (such as *Triton*, an air travel planning assistant described by Davies et al., 2001; or the COLLAGEN *Email Assistant* described by Lesh, Rich, & Sidner, 1999). COLLAGEN uses the generic recipes specified by the system developer to interpret the user's actions and decompose the user's goals into subgoals.

Rickel et al. (2002) describe how Paco uses straightforward discourse generation rules. Paco represents the procedures it will teach using COLLAGEN's declarative language for domain-specific procedural knowledge. This knowledge serves as a model of how domain tasks should be performed. Each task is associated with one or more recipes (i.e., procedures for performing the task). Each recipe consists of several elements drawn from a relatively standard plan representation. First, it includes a set of steps, each of which is either a primitive action (e.g., press a button) or a composite action (i.e., a subtask). Composite actions give tasks a hierarchical structure. Second, there may be ordering constraints among the steps that define a partial order over them. Third, a task and its steps can have parameters, and a recipe can specify constraints (bindings) among the parameters of a task and its steps. Finally, steps can have preconditions (to allow COLLAGEN to determine whether a step can be performed in the current state) and postconditions (to determine whether the effects of a step have been achieved).

2.2.2 Planning With Deterministic Actions

The classical AI planning paradigm[8] is strongly influenced by systems like GPS, the General Problem Solver (Newell & Simon, 1963), and STRIPS, the Stanford Research Institute Problem Solver (Fikes & Nilsson, 1971).

STRIPS has established a restricted language to describe states and operators, which is still used by many deterministic planners today. States are represented by conjunctions of function-free ground literals, that is, predicates applied to constant symbols, possibly negated. Many planning systems adopt the convention that if a state description does not mention a given positive literal then the literal can be assumed to be false (*closed world assumption*). STRIPS operators consist of three components: (1) the name or description of the action, (2) a list of *preconditions* that have to hold for an action to be applicable, and (3) the effects of the operator in terms of an *add list* and a *delete list*. The basic idea of STRIPS planning is to find a sequence of actions (applications of operators) to turn the initial state representation to a goal state representation.

There exist several extensions of STRIPS, for example, ABSTRIPS (Sacerdoti., 1974), the first hierarchical planning system. *Hierarchical task net (HTN) planning* is addressed in more detail by Sacerdoti (1977). Some of the most popular planners for deterministic actions include Prodigy (Veloso et al., 1995) and UCPOP (Penberthy & Weld, 1992).

Some planning approaches with deterministic actions are based on a special data structure called *plan graph*. A plan graph allows to give better heuristic estimates for the exploration of the state space. IPP (Koehler, 1998) is a planning approach based on Graphplan (Blum & Furst, 1997), which explicitly takes resources into account. It assumes that actions consume or produce resources and thereby is able to guarantee that a plan is compliant with existing resource limitations.

It is beyond the scope of this thesis to mention more than just some of the most important planning approaches developed during the last four decades. A comprehensive survey of the complete field of automated planning is given, for example, by Ghallab, Nau, and Traverso (2004).

[8]We use the terms *planning with deterministic actions* and *classical planning* synonymously in this thesis.

This section reviews systems that adopt one of the planning approaches within the classical AI planning paradigm. They all use some STRIPS-like representation (preconditions and effects) of the planning operators. We consider two examples, Extended TEXPLAN and WIP/PPP, for plan-based approaches to generate multimedia explanations and presentations (similar approaches for the generation of pure textual explanations are described by Moore & Paris, 1993, and Cawsey, 1993), a plan-based elevator controller by Koehler and Schuster (2000), and Writer's Aid, a writer's collaborative assistant.

2.2.2.1 Maybury (1991): Extended TEXPLAN, Planning Multimedia Explanations

Maybury (1991) describes an extension of the text planning system TEXPLAN[9] for the plan-based generation of multimedia explanations of route descriptions. The system can answer queries like "Where is city A?" or "How do I get from A to B?". Extended TEXPLAN is a *hierarchical planner* working on plan operators that describe *communicative acts*. It focuses on the design of dynamic, narrated animations of routes over an object-oriented cartographic information system. Thereby, higher level rhetorical (media-independent) actions (such as narrate, explain, argue, compare, and so forth) can be composed of one another or of more primitive speech acts. Similarly multimedia communication can be composed of, for example, linguistic, graphical, or physical (gestures) acts that, appropriately coordinated, can perform a communicative goal. The plan operators defining the communicative acts are encoded in an extension of first order predicate calculus.

2.2.2.2 Wahlster et al. (1993): WIP, a Multimodal Presentation System

The multimodal presentation system WIP[10] (Wahlster, André, Finkler, Profitlich, & Rist, 1993) allows the generation of alternate presentations of the same content taking into account various contextual factors. WIP is a highly adaptive interface since all of its output is generated on the fly and customized for the intended target audience and situation. All presentation decisions are postponed until runtime. WIP does not use any predesigned texts or graphics, that is, each presentation is designed from scratch by reasoning from first principles. The system was tested in different application domains like generating illustrated explanations and instructions on using an espresso machine, assembling a lawn-mower, and maintaining a modem. A basic principle underlying the WIP model is that the various constituents of a multimodal presentation should be generated from a common representation of what is to be conveyed. The system coordinates output in different modes, enforcing a consistent, harmonious, and esthetic integration of the output's multimodal constituents. WIP plans a multimodal presentation incrementally to be able to respond to unexpected events promptly. This means that many elements of the presentation are determined not long before they are actually output (cf. Section 2.2.2.3).

In the WIP system, content and mode selection are interleaved by the use of a uniform planning mechanism. Presentation planning and content realization are performed by separate components. This modularization enables parallel processing, but requires the communication between single components. WIP's planning module (RAT[11]) is implemented as an extension of the terminological logic KRIS[12] (Baader & Hollunder, 1991) to combine reasoning about concepts and instances

[9]TEXPLAN stands for **T**extual **EX**planation **PLAN**ner.

[10]WIP stands for **W**issensbasierte **I**nformations**p**r¨asentation *[knowledge based information presentation]*.

[11]RAT stands for **R**epresentation of **A**ctions in **T**erminological Logics.

[12]KRIS stands for **K**nowledge **R**epresentation and **I**nference **S**ystem.

of the domain as well as actions, plans, and relations between them. In contrast to the system developed in the follow-up project PPP, WIP does not allow for much interactivity. We come back to the WIP system in Section 4.1.1.1 in the context of external assistance systems for technical devices.

2.2.2.3 André et al. (1996): PPP, the Personalized Plan-Based Presenter

PPP, the Personalized Plan-Based Presenter (André, M¨uller, & Rist, 1996), generates multimedia presentations that are presented by a life-like character, the so-called PPP Persona. Similar to WIP, PPP can be used for the automated generation of instructions for technical devices or for the presentation of predesigned material retrieved from the web. The PPP Persona shows, explains, and verbally comments textual and graphical output on a window-based interface, and it can use a pointing stick to refer to graphical objects or textual information on the display. It responds to follow-up questions concerning the domain as well as to meta comments on the act of presentation.

Using a life-like character requires temporal coordination of different media. The principles that underly coherent text-picture combinations in the PPP system are based on (a) the generalization of speech act theory to the broader context of communication with multiple media, and (b) the extension of rhetorical structure theory to capture relations that occur not only between presentation parts realized within a particular medium but also between parts conveyed by different media.

When designing a presentation, PPP employs PrePlan, a presentation planner based on the work of André (1995), to build up a temporal constraint network. The temporal constraint network is checked for consistency by computing the numeric ranges on each interval endpoint, the difference endpoints, and all so-called Allen relationships[13] between each pair of intervals (cf. Rist, André, & M¨uller, 1997). Then, a partial schedule is constructed by resolving all disjoint temporal relationships between intervals and computing a total temporal order. Since the temporal behavior of presentation acts may be unpredictable at design time, the schedule is refined at runtime by adding new metric constraints to the constraint network. This behavior could with some respect also be considered as interleaving planning and execution (cf. Section 2.2.3).

2.2.2.4 Küpper & Kobsa (1999, 2001): User Tailored Plan Generation and Presentation

K¨upper and Kobsa (1999, 2001) regard the output of an advice-giving system as a plan to be executed by the user. They present an approach for *advice-giving in online help and assistance systems* employing several off-the-shelf tools in a two-stage process: (1) The plan generation process (K¨upper & Kobsa, 1999) uses the partial order planner UCPOP (Penberthy & Weld, 1992) to generate plans that take the capabilities of the users into account. The underlying user model specifying the user's capabilities are represented with the user modeling shell BGP-MS (Kobsa & Pohl, 1995). (2) The plan presentation process (K¨upper & Kobsa, 2001) determines what knowledge must be provided to ascertain that the user understands the plan and is able to perform it. Unexpected problems during plan execution may involve replanning (i.e., the plan generation process and the plan presentation process are interleaved). Two peculiarities of the approach are: (a) It provides explanations for the plan steps, and (b) the plan generation process can be biased such that it prefers plans that contain as little information unfamiliar to the user as possible.

[13]Allen (1983) introduces an interval-based temporal logic together with a computationally effective reasoning algorithm based on constraint propagation.

2.2.2.5 Köhler & Schuster (2000): *Elevator Control* as a Planning Problem

Koehler and Schuster (2000) have developed a plan-based elevator controller that serves as a basis for the elevator control software in Schindler elevators. The plan-based approach allows for new services, such as access restrictions, VIP service, nonstop travel, and the like. The system presupposes a so-called destination control, that is, passengers input their destination before they enter the elevator. After input of the destination, the elevator control system selects an elevator for the transport of the passenger—it is assumed that more than one elevator is available. The selected elevator meets all requirements (i.e., it cares for all demanded services) and offers the fastest and most comfortable ride. Passengers with identical destinations can be grouped together and can thereby reach their destination faster and more comfortable than otherwise. The system uses a domain-independent planning approach based on the PDDL[14] planning formalism (Ghallab et al., 1998).

2.2.2.6 Baus et al. (2002): REAL, a Resource-Adaptive Pedestrian Navigation System

Baus, Kr¨uger, and Wahlster (2002) describe a resource-adaptive mobile pedestrian navigation system developed in the READY partner project REAL. The system applies different technologies to detect the user's position (e.g., GPS or GSM/UMTS for outdoor positioning, infrared or bluetooth for indoor positioning) and adapts the multimodal presentation of route descriptions and navigation recommendations according to the detected limited technical and—if provided[15]—cognitive resources. The REAL system combines presentations on a stationary information kiosk with output on mobile devices like PDAs or glasses with clip-on-display. Forms of presentation range from static and dynamic sketch-like graphics to birds-eye or user-perspective walk-throughs.

REAL uses the presentation planner PrePlan (André, 1995), that was also the basis for planning techniques applied in the PPP project (cf. Section 2.2.2.3). The main idea is to decompose a goal until no others but primitive actions (simple presentation steps) remain to be executed.

For the generation of adaptive route recommendations, REAL also uses the method described by Bohnenberger and Jameson (2001). Moreover, Bohnenberger and Kr¨uger (2001) describe an approach that allows to combine predefined routes with decision-theoretic planning. Providing a predefined route can be useful for different reasons, for example, to guide a user on a route that he is already familiar with. The motivation for a combination is that, by the decision-theoretic planning process, the predefined routes can be augmented to provide good recommendations after the user has taken a wrong turn. In Section 4.3, we describe this approach in detail.

2.2.2.7 Babaian et al. (2002): *Writer's Aid*, a Writer's Collaborative Assistant

Babaian, Grosz, and Shieber (2002) describe Writer's Aid, a system that deploys AI planning techniques to help an author identifying and inserting citation keys. The system autonomously generates and executes plans for finding and caching potentially relevant papers and their associated bibliographic information from various sources on the Internet. The system identifies the user's preferred bibliographies, his own bibtex files, and online scientific collections. While working with the system, the user, instead of citing a reference explicitly, specifies some search parameters: keywords and an indication whether he wants only the bibliographic data to be retrieved or

[14]PDDL stands for **P**lanning **D**omain **D**efi nition **L**anguage.

[15]An assessment of the cognitive resources can be provided, for example, by the user modeling component of the READY prototype.

also an electronic version of the paper itself. Writer's Aid processes the requests in a round-robin[16] fashion, gradually increasing the maximum complexity level of finding and executing the solution plan. To implement the gradual increase of solution complexity, Writer's Aid performs iterative deepening in hypotheticals, that is, in partial plans that hypothesize on the value of an unknown proposition (e.g., the location from where a paper might be available) or subgoal (e.g., how to find out about the location of a paper). By verifying a hypothesis via execution of a sensing action, the planner eventually is able to either find a plan or determine that the goal is unsatisfiable.

2.2.2.8 Applicability and Limitations

As Extended TEXPLAN and WIP/PPP illustrate, classical planning is a powerful means for systems to generate well coordinated output, for example, for multimedia presentations. Plans can be used to provide different presentations for different contexts, but the context has to be specified right from the beginning in terms of the initial state from which the planner starts. Changing contexts cannot be considered during the execution of a plan. Moreover, this approach cannot take the user's reaction to the presented output into consideration—taking the user's feedback into account requires at least some kind of conditional planning (see Section 2.2.4). In unconditional planning, a plan is used to specify a sequence of actions to be applied in an unobservable environment. Therefore, a plan cannot take future events that are initiated by anyone else but the plan executor into account. For example, the plan-based elevator controller (Section 2.2.2.5) cannot consider future passengers arriving at the elevator. Recent work in the same domain employing decision-theoretic planning (Nikovski & Brand, 2003) can take exactly this uncertainty into account. Writer's Aid faces a similar problem and uses a trick to deal with the uncertainty in its environment (intranet and Internet): It repeatedly generates new plans (in a clever way) until one plan finally succeeds. This approach works for the planning of the system's interaction with the sources of the intra- and Internet but would not work for the planning of the interaction with a human. The user would soon be tired of a system that repeatedly confronts him with plans in a trial-and-error manner. The next two sections describe methods to deal qualitatively (i.e., not requiring knowledge of probabilities) with uncertainty about future events.

2.2.3 Interleaving of Planning and Execution

As an alternative to quantitatively modeling uncertainties about a process to be planned, planning and execution of plan steps can be interleaved (see, e.g., Pryor & Collins, 1996). The idea is simply to construct a plan that is fully specified up to the information-gathering step, then execute the plan to that stage, and construct the rest (or the next part) of the plan once the information has been gathered. Interleaving planning and execution in this way has the advantage that it avoids planning for contingencies that do not actually arise. However, it loses some of the advantages of planning in advance, for example, possible interference between actions performed before and after the information gathering might be missed, meaning that the planner might find a suboptimal plan. The complexity evolving from the consideration of uncertainty is avoided at the price that the result of the whole process to be planned cannot be foreseen entirely.

[16] In the given context, round-robin means: Try a plan with some hypothesized propositions or subgoals; if it does not succeed, try a more complex plan, possibly with different hypotheses.

2.2.3.1 Wahlster et al. (2001): *SmartKom*, a Multimodal Dialog System

Wahlster, Reithinger, and Blocher (2001) and Wahlster (2002) describe the multimodal dialog system SmartKom, which combines speech, gesture, and mimics input and output. The system supports situated understanding of possibly imprecise, ambiguous, or partially multimodal input, and the generation of coordinated, cohesive, and coherent multimodal presentations. SmartKom merges three user interface paradigms (spoken dialogs, graphical interfaces, and gestural interactions) to achieve multimodal interaction. Like PPP (cf. Section 2.2.2.3), the system employs an agent visualized as a life-like character to which the user can delegate a task. The agent elicits the specification of the delegated task in a collaborative dialog and elaborates a plan to achieve the user's intentional goal.

There are several subsystems of SmartKom that do planning. The main planner, which is domain independent, is both responsible for the dialog planning and for the coordination of SmartKom's actions, for example, the synchronization of devices. It is a hierarchical approach in that, during the execution of a plan, it can invoke subordinate planning processes to achieve subgoals. That is, the user can always interrupt the system during a task. For example, while the system gives a route description, the user might ask "What is the building over there?". In such a case, the system stops the execution of the current plan, generates a plan for the new subgoal, and after achieving the subgoal continues with the execution of the superordinate plan. In this sense, the main planner can also be considered as a hierarchical planner. It is based on a STRIPS-like planning language. A text generator, a presentation generator, and a dynamic help module employ individual subordinate planners, for example, PrePlan (André, 1995), which has already been used in PPP (Section 2.2.2.3) and REAL (Section 2.2.2.6) for the generation of (multimodal) presentations.

2.2.4 Contingency/Conditional Planning

The term contingency (or conditional) plan[17] refers to a plan that contains actions that may or may not actually be executed, depending on the circumstances that hold at the time of the execution of a plan (Pryor & Collins, 1996). Contingency plans involve alternative courses of action to be performed in different circumstances. Contingency planners are related to the method of interleaving planning and action in that they cannot use information about probabilities, but they can construct plans in circumstances in which no such information is available. They differ from planners that interleave planning and execution in that contingency plans cover the whole process to be planned, while interleaving planning and actions gets a system only as far as no additional information is required. Contingent planners merely decide at runtime which branch of the planning tree to select. But no branches have to be constructed in addition to those that already exist as more information becomes available. The PHI system, which we have introduced in Section 2.2.1.1, uses conditional planning.

2.2.5 Global Anticipation Feedback

Ndiaye (1998) introduces the concept of *global anticipation feedback (GAF)*, a strategy that allows dialog participants to predict the responses of their dialog partners by hypothetically assuming the

[17]The terms *contingency planning* and *conditional planning* are often used synonymously in literature. For an introduction, see, for example, Russell & Norvig, 2003, Part IV, Chapter 12.4.

partner's role. The technique was developed to extend the reasoning capabilities of the PRACMA system (Section 2.1.2.2) and carries on the idea of *local anticipation feedback* as it is described by Jameson (1989) in the context of the Imp system. The GAF approach allows the system to take either roles, buyer or seller. To anticipate several steps of a dialog, the system alternatingly takes the role of the buyer and the role of the seller in a recursive way. That is, the system takes up the perspective of the user and in effect invokes itself as a subroutine. The action that its own instance generates is the predicted response of the user. This way, the system implicitly generates a decision tree with all potential actions of the system and the reactions of the user in turn (cf. Ndiaye, 1998, Chapter 7) and propagates utilities from the leave nodes to the root node, resembling the *minimax algorithm* used for board games. The method of Ndiaye focuses largely on the aspect of mutual invocation of the same module taking different roles. The branching factor of the corresponding decision tree is enormous, such that the look ahead usually has to be restricted to a small number of steps of steps. Clever strategies for the exploration and evaluation of the decision tree contribute to a slight relaxation of this problem.

2.2.6 Reinforcement Learning

Reinforcement learning (see, e.g., Kaelbling, Littman, & Moore, 1996, for a survey) is a method to learn a strategy solving a problem through trial-and-error interactions with a dynamic environment. It is a way to achieve a desired behavior of a system without explicitly specifying how a task is to be achieved. The system learns from indirect and delayed rewards to choose the desired behavior in terms of the sequences of actions that produce the greatest cumulative reward. Reinforcement learning and decision-theoretic planning are based on the same mathematical framework. A decision-theoretic planner constructs a policy[18] (i.e., a strategy to accomplish a task) on the basis of a process model specified at design time, while a reinforcement learner learns a policy via the reinforcement it gets in terms of rewards for the actions that it selects. This means that reinforcement learning is especially suitable if the interrelations between actions and effects are not clear at design time—or too complex to be specified manually. On the other hand, a system using pure reinforcement learning will usually show a poor initial behavior—unless the system is trained beforehand. Yet, training may require a huge amount of sample cases, depending on the complexity of the domain. Reinforcement learning has successfully be applied to, for example, robot control problems, optimization problems in factories, or playing board games. In the context of user-adaptive systems, we will in the following discuss ELVIS, a spoken dialog front end for email systems, ATIS, a prototypical air travel information system, and WebWatcher, an assistant for the World Wide Web.

2.2.6.1 Walker (2000): ELVIS, the Email Voice Interactive System

ELVIS[19] is a spoken dialogue system that supports access to email over the phone (Walker, 2000). ELVIS is able to learn from its experience with other users what information to communicate, as well as how and when to communicate it. It can select from a set of strategies to summarize email folders. A summary can consist of a simple statement about the number of messages in different folders ("You have five new messages.") or it can provide more details (e.g., sender and subject) about the messages in a particular folder.

[18]The technical term *policy* is defi ned in Section3.1.4.

[19]ELVIS stands for **E**mai**L** **V**oice **I**nteractive **S**ystem.

ELVIS applies Q-learning to determine the optimal summarization strategy. Q-learning is a form of reinforcement learning. It is called a model-free method because it does not require a model for either learning or action selection (see, e.g., Russell & Norvig, 2003, Part VI, Chapter 21, for an introduction to reinforcement learning; Q-learning is explained in Chapter 21.3). As a prerequisite for this approach to work, the system needs a performance function for assigning a utility to the final state of a dialogue. ELVIS uses the PARADISE evaluation framework (Walker, J., Kamm, & Abella, 1997) for this purpose.

2.2.6.2 Levin et al. (2000): ATIS, an Air Travel Information System

Levin, Pieraccini, and Eckert (2000) describe a stochastic model for dialog systems based on Markov decision processes. They characterize the problem of finding a dialog strategy as optimization problem and show how to solve this problem by reinforcement learning. Their approach is illustrated by example dialogs in the context of the air travel information system ATIS. The system's goal is to steer the dialog into a direction such that the user's query is constrained in a way that a reasonable amount of appropriate flights can be fetched from the airline database in the end. Levin et al. (2000) show that a system which starts without any initial knowledge eventually converges to a very reasonable dialog strategy. Yet, convergence can already take a large amount of training cases for strategies to accomplish rather simple tasks.

2.2.6.3 Armstrong et al. (1995): *WebWatcher*, a Tour Guide for the Web

WebWatcher (Armstrong, Freitag, Joachims, & Mitchell, 1995) is a tour guide software agent for assisting users browsing the world wide web. WebWatcher accompanies users from page to page and recommends hyperlinks that it expects to be of interest for the user (cf. Letizia, Section 2.1.2.3). WebWatcher gains experience from the interaction with its users. It can learn that two concepts are related to each other even though they share no words in common and might be connected only by a sequence of hyperlinks.

WebWatcher's task is to recommend an appropriate hyperlink given an interest (which the user describes in terms of keywords at the beginning of each session) and a Web page. The quality of a hyperlink is interpreted as the probability that a user will select this link given the current page and the user's interest. WebWatcher combines two approaches (see also Joachims, Freitag, & Mitchell, 1996, 1997) to learn the quality of a hyperlink: (1) It uses previous tours (of other users) to augment the internal representation of the links contained in a page with several keyword lists that relate to possible interests; the link quality for each hyperlink is estimated to be the average similarity of the k (usually 5) highest ranked keyword sets associated with the link; a hyperlink is suggested if its quality exceeds a threshold; for each page, up to three hyperlinks can be recommended. (2) It tries to find tours through the web that maximize the amount of relevant information over the complete trajectory. In general, a user can select a hyperlink recommended or not. Each selection of a hyperlink is used as a training example for learning to improve future advice. Learning is accomplished by annotating each hyperlink with the interests (keyword lists) of the users who took this hyperlink on tours with positive feedback. Thus, whenever a user follows a hyperlink on a successful tour, the description of this link is augmented with the keyword list the user typed in at the beginning of the tour. The initial description of a link is the underlined text.

2.2.6.4 Applicability and Limitations

Reinforcement learning is a comfortable approach to learn strategies (policies) for sequential decision processes that are inaccessible or too complex to be modeled by hand. The major drawback of the approach is the necessity of a preparatory training phase before a system can be used in practice. This training phase can be—dependant on the size of the domain—extensive and time-consuming, as indicated in the context of the ATIS system (Section 2.2.6.2). Some domains require systems that can hardly be trained other that in operation with "real" users—users that actually prefer to benefit from the system and not vice versa. In some domains, for example, recommender systems for the World Wide Web (cf. Section 2.2.7.1), it would be completely unacceptable to train the system with real users.

As the examples reviewed in this section suggest, reinforcement learning is nevertheless useful for user-adaptive systems. In some domains, a large amount of training examples can easily be acquired or is already available, for example, in terms of logs of the system's earlier interactions with the current or with other users. Sometimes, the system can learn unnoticedly during interactions with the user. For example, in domains where the user might just ignore poor initial recommendations, while he gets to appreciate the recommendations as they improve over time. In such a case, it is important that the user is not dependent on the system's recommendations. Or more general, that the user is always in control, that is, that he can choose if he makes use of what the system offers to him or not.

2.2.7 Decision-Theoretic Planning

Decision-theoretic planning is recognized as an attractive extension to the classical AI planning paradigm. It found its way into AI through applications in robot control. By its mathematical foundation, decision-theoretic planning provides a neat approach to deal precisely with the uncertainties of a domain. The solution to a decision-theoretic planning problem is a *policy* (a mapping from states to actions, cf. Section 3.1.4). There are two classes of decision-theoretic planning problems: fully observable Markov decision problems, in which only the uncertainties about the results of actions are considered, and partially observable Markov decision problems, which additionally take uncertainty about the current state into account. We give a detailed introduction to the formal framework of Markov decision processes in Chapter 3. User-adaptive systems that use decision-theoretic planning are still rare. Due to the close cooperation with the READY project, the decision-theoretic planning methods described in this thesis have also been applied in the REAL project for the generation of adaptive route recommendations. Other successful applications of decision-theoretic planning are the MDP-based recommender system for the Mitos book store and the spoken dialog front end of the nursebot *Florence Nightingale*.

2.2.7.1 Brafman et al. (2003): MDP-Based Recommender System for the *Mitos* Book Store

Brafman, Heckerman, and Shani (2003) introduce an MDP-based recommender system that is used in the Mitos online book store. To the best of our knowledge, it is the first commercially used recommender system based on decision-theoretic planning. Standard reinforcement-learning techniques are inadequate for the use in online recommender systems because a period of random recommendation during training is unacceptable for a commercial site. Therefore, historical data describing user behavior is exploited to learn a stochastic model, which forms the basis for the initialization of the MDP. The stochastic model provides the probability that the user will purchase a particular item given his sequence of past purchases. The states of the MDP are composed of a k-tuple specifying the items purchased so far. The system's actions correspond to a recommendation of an item. After each recommendation, the user can (1) accept the recommendation and purchase the recommended item, (2) select some non-recommended item, or (3) select nothing. A recommendation increases the probability that a user will buy an item (see also Brafman, Heckerman, & Shani, 2004, for a more detailed description of the approach).

The optimal policy takes the likelihood of a recommendation to be accepted by the user into account. Using decision-theoretic planning allows the system to maximize the expected utility of its recommendations (e.g., the system can recommend a product that has a slightly lower probability of being bought, but generates higher profits) and take future consequences of its recommendations into account (e.g., the system suggests an item whose immediate reward is lower, but leads to more likely or more profitable rewards in the future). The approach includes several approximation techniques to cope with the huge size of the state space. It exploits the special structure inherent to the domain in several ways and thereby prunes the state-space such that a standard algorithm (policy iteration) can be employed as a tractable solution method.

2.2.7.2 Roy et al. (2000): *Florence Nightingale*, Spoken Dialog Management

Roy, Pineau, and Thrun (2000) describe a spoken dialog manager for the nursebot *Florence Nightingale (Flo)*. *Flo's* job is to assist elderly people with their daily tasks. Users can inquire, for example, about the time, the patient's medication schedule, or what is on different TV stations, and they can select from a range of robot motion commands (e.g., move to the kitchen). The system uses POMDPs (see Section 3.1.6) to generate dialog strategies. Thereby, the state represents the user's intentions (rather than the system state), which are only partially observable from the perspective of the system. The system can make inferences about the user's intentions only from the observations about the user (in this case from the speech input). *Flo's* dialog management is robust with regard to noisy conditions: It adjusts the dialog policy as the quality of speech recognition degrades and makes fewer mistakes if compared to an FOMDP (see Section 3.1) approach for the same purpose.

The computations for an exact solution of such a real world dialog scenario modeled as POMDP would be computationally intractable. *Flo* therefore uses an algorithm based on augmented MDPs (Roy & Thrun, 1999), which approximates solutions for POMDP-style problems. Augmented MDPs base on the assumptions that the most likely state and the *entropy* of the belief state together approximate a sufficient statistics for the actual belief state.

2.3 Discussion

In this chapter, we reviewed important approaches for decision making in user-adaptive systems and a variety of systems to exemplify these approaches. We distinguished two main classes of systems: systems that make individual decisions and systems that use some kind of planning for the interaction with the user.

Among the systems that confine themselves to making individual decisions, we started with those that observe and imitate the user. This approach is well described by the term "looking the user over the shoulder", which implies that systems following this approach usually have the ability to learn from examples and to improve over time. The ability to learn, which is a very desirable property in principle, is connected with the problem that a system's initial performance can be poor if the system learns its behavior from scratch. Additional competence acquisition schemes, as described for Maxims, can be employed to overcome this problem. The purpose of looking the user over the shoulder is usually that the system tries to automate certain tasks. These tasks are often not very difficult but rather tedious to the user because of their monotonous nature. The approach allows to predict the user's next action and adopt it as the system's recommendation. Scalability is usually not a critical aspect of systems that imitate the user.

Next, we considered systems that rate alternatives in terms of their relevance with respect to the user's needs. These systems (explicitely or implicitly) determine a preference order among the available options and are dedicated to make recommendations or provide some sort of guidance. As Letizia illustrates, a preference order can also be learned solely on the basis of examples. A neat mathematical framework to model a user's preferences provides the multi-attribute utility theory.

Decision-theoretic methods for making individual decisions are based on the principle of maximal expected utility (MEU). Making a decision according to the MEU principle is considered rational. The basic tools for making rational decisions are Bayesian networks, influence diagrams, dynamic Bayesian networks, and dynamic decision diagrams. Besides READY, several projects in the Microsoft Decision Theory & Adaptive Systems Group (DTAS)[20]—starting with the Lumière project—and the *DT Tutor* system have used these methods for the purposes of generally assisting the user, automating tasks, giving recommendations, grounding[21], and tutoring. Decision-theoretic methods are most applicable for selecting an action from a small number of options (e.g., selecting a form of output presentation or selecting the next tutorial task) and under the condition that the decision whether to perform an action is a yes-no-decision (e.g., offering to automate a task or phrasing a question to solicit some information). The ability to learn is not inherent to decision-theoretic methods, but an automatic fine tuning of probabilities can be incorporated by processing the relative frequencies of the outcomes of relevant uncertain actions. Scalability can be a problem in the context of dynamic Bayesian networks and dynamic decision networks, respectively. There exist approximative and exact methods to overcome the complexity problems (see, e.g., Kjærulff, 1995; Boyen & Koller, 1998; Brandherm, 2003).

Systems that make only individual decision do not question what the influence of the current decision on forthcoming situations is. Taking a decision's future consequences into account requires optimizing the result of a sequence of actions or—considering that the results of individual actions can be uncertain—optimizing the result of all possible sequences of actions. We reviewed examples of different planning approaches used in user-adaptive systems.

As a first approach to decide about complete sequences of actions, we considered plan libraries. This approach is often too static, inflexible, and problematic with regard to scalability. In particular, the reuse of a plan can require much effort to adapt an old plan to a new situation, such that planning from scratch is sometimes faster. Nevertheless, the approach has successfully been applied in assistance systems and tutoring systems. Its main advantage is the potential reuse of plans in domains where reuse works effectively.

There exist a lot of user-adaptive systems that employ planning under the assumptions that the considered action are deterministic. Such planning was applied, for example, for the presentation of route descriptions or operating instructions. Among others, several projects in the Intelligent

[20]Information on the Microsoft Decision Theory & Adaptive Systems Group can be found at `http://research.microsoft.com/research/dtg/default.asp`.

[21]Grounding: the process of establishing, maintaining, and confi rming mutual understanding.

User Interface Lab at DFKI[22] have used planners for deterministic actions. Various techniques have been developed for planning with deterministic actions. The hierarchical nature of many planning problems yielded hierarchical planners, as for example used in TEXPLAN or SmartKom. A common problem of planning with deterministic actions is: What can the system do if a plan fails? In this case, a system can either try to repair the current plan or to construct a new plan from scratch. Both can be time-consuming and is therefore not practical for domains in which a quick response is important and expected from a system. Moreover, the need to repair a plan or to do replanning already implies that the original plan did not account for all relevant eventualities, that is, some aspects about the process to be planned have not been considered in the original plan. The resulting successive execution of the two plans can be suboptimal. In fact, not all eventualities can be foreseen. However, appropriate approaches exist to consider those that can be foreseen.

A simple approach to deal with information that becomes available only during the execution of a plan is interleaving planning and execution, as for example the SmartKom system does. Unfortunately, the approach has the same major drawback as replanning or repairing a plan: The process to be planned is possibly not planned ahead to the extend that it could be in principle. Future consequences of decisions can be considered only up to the next information gathering step. The overall result can be suboptimal.

Conditional planning allows to plan a process in which some information becomes available only during the execution of a plan to the very end. Conditional plans are not just sequences of actions, but they have a tree structure: The branch of the tree is selected according to the information that becomes available at the corresponding plan step. The ability to incorporate feedback from the user does not yet enable the system to *steer* the interaction into a favorable direction for the user. If several options are available (several outcomes of the user's actions are possible, respectively), the approach considers all possibilities to be equally probable. Moreover, the goal of a planning process sometimes consists of several subgoals (e.g., buying several items from a shopping list), and the achievement of these subgoals might have to be traded off against a competing goal (e.g., finishing a shopping task as fast as possible). To deal with such problems, quantitative considerations as expressed by rewards for goals, costs for actions, and probabilities for the outcomes of actions are indispensable.

The global anticipation feedback approach by Ndiaye (1998) describes a step towards decision-theoretic planning. The system repeatedly makes individual decisions, alternatingly taking the role of both the system and the user. However, because of an enormous branching factor of the corresponding decision tree, the scalability of this approach is problematic: It can be used to look ahead only some but not many steps.

A very promising approach to globally optimize a process with quantitatively known uncertainties is reinforcement learning. We already emphasized its close relation to decision-theoretic planning by the same underlying mathematical framework. Reinforcement learning has already successfully been applied in user-adaptive systems. But for complex processes, the training phase required by a reinforcement learner can be very time-consuming—which is in general not acceptable for systems that can be trained only by real users. Apart from this restriction, systems using reinforcement learning have the same advantages that claim to be beneficial for the application of decision-theoretic planning in user-adaptive systems.

Decision-theoretic planning is an approach on a well-founded mathematical basis that com-

[22]Information about DFKI, Deutsches Forschungszentrum für Künstliche Intelligenz [German Research Center for Artificial Intelligence] can be found at `http://www.dfki.de/`.

bines probability and utility theory. It allows to take the uncertainty about the outcome of actions quantitatively into account and to trade off the utility of a goal with the costs for the actions to achieve the goal. The desirability of a goal can be quantified, and several goals can be considered in parallel. The MDP-recommender system for the Mitos book store illustrates how decision-theoretic planning enables a system to take the future consequences of its decisions into account. The interaction with the user can thereby be steered into a favorable direction. Decision-theoretic planning also has some drawbacks: It is sometimes difficult to provide probabilities and to come up with appropriate measures for costs and rewards. Where partially observable Markov decision processes are involved, it is in particular the scalability problem that restricts the applicability of the decision-theoretic planning approach. Approximation techniques, as for example described for *Florence Nightingale*, allow at least to consider some small problems relevant in practice.

The possible degree of look ahead of a system taking the future consequences of its decisions into account is important for the quality of its adaptive behavior. Only the *DT Tutor* system, Ndiaye's *global anticipation feedback* approach, and the systems using reinforcement learning or decision-theoretic planning look more than a single step of the interaction with the user ahead. The systems that make individual decisions either predict the user's next action and adopt it as recommendation, or they deduce their next recommendation according to the user's anticipated interests. As for the systems that use recipes to control the interaction, it is rather the designer of the recipes that looks ahead. Systems that use planning with deterministic actions or interleave planning and execution restrict their look ahead to their own actions and do not adequately take the user's actions in their decision making into account.

Some of the systems that we reviewed in this chapter are able to give explanations of their decisions. The systems based on the COLLAGEN architecture can provide static explanations as far as they are predetermined in corresponding recipes. Letizia and the general approach of Kobsa and K¨upper provide at least rudimental explanations of how they arrive at decisions. The term *explanation* was also used in conjunction with Extended TEXPLAN, WIP, and PPP. Note that these systems indeed explain the generated multimedia contents. But they do not provide explanations of how they arrive at a decision. The fact that not many systems provide explanations does not mean that the underlying approaches for decision making do not allow for explanations in principle. The effort to equip a system with an explanation component can vary widely. Presumably, researchers who develop a system often simply refrain from taking this effort because the ability to explain decisions is not in the focus of their research. However, if a system is to be applied in practice, the ability to explain decisions and to answer metacommunicative questions (e.g., why has the system given a particular recommendation?) can enhance the users' acceptance of a system remarkably. We discuss our approach to provide explanations for decision-theoretically planned decisions in the context of user-adaptive systems in Section 7.4.

Eventually, scalability is in general a crucial requirement for an approach to be applicable in practice. Not all of the approaches that we discussed in this chapter scale well. Unfortunately, this in general applies also for decision-theoretic planning. However, in some realistic domains, scalability problems can be overcome by approximative methods.

Tables 2.1 and 2.2 compare some general pros and cons of the approaches and systems discussed in this chapter with the goal-priorized decision-theoretic planning approach for a navigation assistance scenario, which we will explain in Chapter 5.

System	Domain & Purpose	Decision Method	FC	UC	MG	RL	TI
Maes (1994): *Maxims*	Electronic mail assistance; automation of incoming email processing	comparison with memorized situations	–	–	–	–	–
Cypher (1991): *Eager*	Repetitive tasks anticipation; task completion and automation of tasks in the HyperCard environment	programming by example	–	–	–	–	–
Jameson et al. (1995): PRACMA	Dialog system for purchasing a car; recommendations	MAUT, preference order	(+)	+	–	+	+
Lieberman (1995): *Letizia*	WWW browsing assistance; guidance, recommendations	examples, history, preference order	–	–	–	(+)	–
Shearin and Lieberman (2001): *Apt Decision*	Rental real estate recommendations	examples, history, preference order	–	–	(+)	–	–
Horvitz et al. (1998): *Lumière* and *Microsoft Offi ce Assistant*	Offi ce applications; automation, assistance	BNs, influence diagrams	–	+	–	–	+
Paek and Horvitz (2000): *MS Presenter* and *MS Bayesian Receptionist*	Offi ce applications; grounding, recommendations	DBNs, DDNs	–	+	–	–	+
Murray et al. (2001): *DT Tutor*	Tutor system; selecting tutorial discourse actions	DDNs	–	+	+	–	+
Rickel et al. (2002): *Paco*	Simulation-based training system; tutoring	plan libraries/recipes	–	–	–	–	+
Davies et al. (2001): *Triton*	Air travel planning assistance	plan libraries/recipes	–	–	–	–	+
Lesh et al. (1999): COLLAGEN *Email Assistant*	Email assistance	plan libraries/recipes	–	–	–	–	+
Bauer et al. (1993): PHI	Logic-based tool for interactive help in a UNIX mail system; assistance, anticipation	planning from fi rst and second principles	–	–	–	–	+
Maybury (1991): TEXPLAN	Multimedia route explanations; guidance, presentation	planning with deterministic actions	–	–	–	–	+
Wahlster et al. (1993): WIP	Generating illustrated explanations and instructions; assistance, presentation	planning with deterministic actions	–	–	–	–	+
Andr´e et al. (1996): PPP	Multimedia presentations with life-like character; assistance, presentation	PrePlan, planning with deterministic actions	–	–	–	–	+

Table 2.1*: Approaches for decision making in user-adaptive systems—part I.*
(The table describes the domain, purpose, and main method of decision making of the systems discussed in this chapter. Moreover, it compares the following characteristics: **FC***: Can $\mathcal{S}$ adequately take the future consequences of its decisions into account?* **UC***: Can $\mathcal{S}$ take uncertainty about the possible outcome of actions into account?* **MG***: Can $\mathcal{S}$ trade off multiple goals?* **RL***: Can $\mathcal{S}$ adapt to resource limitations? And* **TI***: Is the performance of $\mathcal{S}$ training-independent?)*

System	Domain & Purpose	Decision Method	FC	UC	MG	RL	TI
K¨upper and Kobsa (1999, 2001): *Online help systems in general*	Advice-giving in online help systems; assistance	planning with deterministic actions	–	–	–	(+)	+
Koehler and Schuster (2000): *Elevator control*	Plan-based elevator control	planning with deterministic actions	–	–	–	–	+
Baus et al. (2002): REAL	Adaptive route descriptions; assistance, presentation	PrePlan, planning with deterministic actions	–	–	–	(+)	+
Babaian et al. (2002): *Writer's Aid*	Identifying and inserting citation keys; planning information retrieval	planning with deterministic actions	–	–	–	–	+
Wahlster et al. (2001): *SmartKom*	Multimodal dialog system; assistance, guidance, presentation	Interleaving planning and action, hierarchical planning	–	–	–	–	+
Ndiaye (1998): GAF in PRACMA	Sales conversation; anticipation, guidance, recommendations	global anticipation feedback	(+)	+	–	+	+
Walker (2000): ELVIS	Email assistance via phone	reinforcement learning	+	+	–	–	–
Levin et al. (2000): ATIS	Air travel information system; information retrieval	reinforcement learning	+	+	–	–	–
Armstrong et al. (1995): *WebWatcher*	WWW browsing assistant; recommendation	reinforcement learning	+	+	–	–	–
Brafman et al. (2003): *Mitos*	Online book recommendation	FOMDPs	+	+	–	–	+
Roy et al. (2000): *Florence Nightingale*	Spoken dialog management for robot control	augmented MDPs	+	+	–	(+)	+
Bohnenberger (2004): GPDTP for the READY *Airport Assistant*	Airport navigation recommendations	goal-priorized DTP	+	+	+	+	+

***Table 2.2**: Approaches for decision making in user-adaptive systems—part II.*

(The table describes the domain, purpose, and main method of decision making of the systems discussed in this chapter. Moreover, it compares the following characteristics: **FC***: Can $\mathcal{S}$ adequately take the future consequences of its decisions into account?* **UC***: Can $\mathcal{S}$ take uncertainty about the possible outcome of actions into account?* **MG***: Can $\mathcal{S}$ trade off multiple goals?* **RL***: Can $\mathcal{S}$ adapt to resource limitations? And* **TI***: Is the performance of $\mathcal{S}$ training-independent?)*

3 Decision-Theoretic Planning

Decision-theoretic planning (DTP) has been noticed as an attractive extension of the classical AI planning paradigm for sequential decision problems under uncertainty (see, e.g., Boutilier et al., 1999; Russell & Norvig, 2003; Ghallab et al., 2004). Decision-theoretic planning allows to determine optimal sequences of actions that have uncertain effects and, if necessary, even if the system has only incomplete information about the world. In the context of user-adaptive systems, incomplete information can relate, for example, to the current situation of the user who interacts with the system.

The key idea of decision-theoretic planning is to represent the planning problem as an optimization problem. Thereby, the planning domain is modeled as a *stochastic system*, that is, the uncertain outcomes of an action are represented by nondeterministic state transitions. Reaching a goal is associated with a reward, and the solution of a decision-theoretic planning problem is a *policy*: roughly speaking, a strategy that specifies the optimal action for each possible situation. The utility function used in decision-theoretic planning can express preferences on the entire execution path of a policy, rather than just on desired final states.

The problems that can be treated with decision-theoretic planning involve systems whose dynamics can be modeled as *stochastic processes*. A stochastic process in the context of user-adaptive systems is a process in which actions with uncertain results change the state of the interaction between a system and its user. In particular, the outcomes of the user's actions are uncertain (reflected by several transitions from the current to possible next states). A stochastic process can be modeled by *states* and *transitions*. Each action in the dialog between the system and the user corresponds to one or several transitions in the stochastic process. Transitions can also correspond to *exogenous events*, that is, events that are beyond the system's control. In Section 1.4, we have explained different perspectives from which a human-computer interaction process can be considered. It would not be wrong to regard the user's actions as exogenous events but, as we have discussed in Section 1.4, we prefer to view the user's actions on an equal footing with the system's actions in a human-computer interaction process.

In this chapter, we will use a simple abstract human-computer interaction example to explain the basic elements of Markov decision processes—the main framework for modeling and solving decision-theoretic planning problems—and to illustrate common methods to solve them.

3.1 Markov Decision Processes

The problem of planning in an accessible, stochastic environment with a known transition model is called a *Markov decision problem*[1]. The stochastic process by which a Markov decision problem can be solved is called *Markov decision process*. The term *Markov* denotes a special property of the considered stochastic processes: The transition probabilities from any given state depend only on the state and not on the state history. This property is called the *Markov property*[2]. Both, the problem and the process are abbreviated as MDP in the literature. Much research on decision-theoretic planning has explicitly adopted the MDP framework as an underlying model. Markov decision processes, as mentioned above, assume an accessible environment and are therefore more precisely called *fully observable Markov decision processes (FOMDP*[3]*)*. Planning in an unaccessible, stochastic environment, that is, an environment in which the decision maker does not have complete knowledge about the current world state, requires *partially observable Markov decision processes (POMDP)*. Methods used for FOMDPs are not directly applicable for POMDPs. We will see that the problems that we consider in the context of user-adaptive systems can to a great extent be treated as fully observable Markov decision problems.

We explain the framework of Markov decision problems with a simple, illustrating example in the context of user-adaptive systems. Thereby, we refer to an experimental setting that captures important, typical aspects of user-adaptive systems. The corresponding experiment[4] was introduced by Jameson, Großmann-Hutter, March, and Rummer (2000) and referred to in different contexts by researchers in the READY project (e.g., Bohnenberger, 2000; Jameson et al., 2001).

3.1.1 Experiment

Jameson et al. (2000) wanted to find out how the performance of a user following spoken instructions of an assistance system (e.g., a computer support hot-line) varies with different instruction strategies. They set up an abstract environment that captures some typical characteristics of a user's interaction with a computer system. The subject had to handle a primary and a secondary task. The primary task consisted of executing sequences of simple spoken instructions. The secondary task was to observe a lamp and press the space bar whenever the lamp repeatedly flashes in the same color during the experiment. The secondary task was supposed to distract the subject while he tried to handle the primary task and expected to increase the mismatches rate.

The question to be answered by this experiment was: Should the system present the instructions rather (a) in a *stepwise* manner, that is, allowing the subject to execute each instruction in the sequence before hearing the next one, or (b) in a *bundled* manner, that is, the subject starts executing the first instruction only after the system has given the complete sequence of instructions. Both the stepwise and the bundled presentation have advantages and disadvantages. Giving the instructions stepwise requires the subject to confirm each execution of an instruction (pressing an okay button), which requires a certain amount of time and effort on the subject's part. The bundled presentation requires the subject to store several instructions in his *working memory*, which may cause a higher error rate for executing the instructions. This should in particular be the case if

[1]Andrei Andreyevich Markov (1856-1922) was a Russian mathematician at Saint Petersburg University, where he began as a professor in 1886. His work launched the theory of stochastic processes.

[2]One can also say: "A stochastic process is Markov" or "The Markov assumption holds for a stochastic process".

[3]We use the term FOMDP only if we want to emphasize the full observability of the corresponding process.

[4]We sometimes refer to this experiment as the instructions experiment.

the subject is distracted by a secondary task. Figure 3.1 summarizes the error rates and execution times for different bundle sizes and situations with and without distraction by the secondary task. The planning of the instruction strategy is supposed to yield a trade-off between minimizing the error rate and minimizing the time needed to complete an instruction sequence (cf. the ATM example explained in Section 1.2).

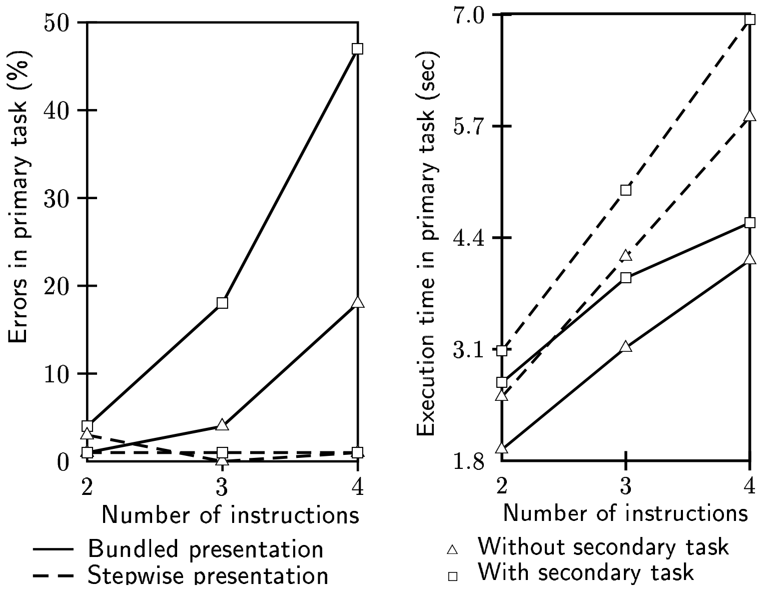
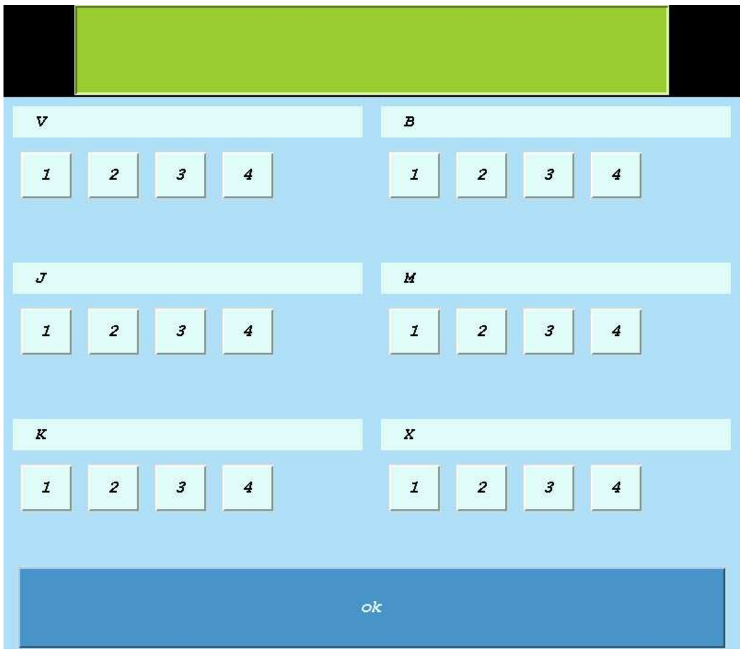


***Figure 3.1**: Error rates and execution times in the experimental setting.*
(The diagram on the left hand side shows that the subjects made remarkably more mistakes with increasing bundle size, especially when they were distracted by the secondary task. The error rate was generally low when instructions were presented stepwise. The diagram on the right hand side indicates the tendency that subjects needed more time for the execution when the instructions were presented stepwise.)

The display for the experiment (Figure 3.2) shows a lamp (used for the secondary task) on top of the screen and a panel for the primary task below it. The panel for the primary task is divided into six regions, each of which is labeled with a letter. There are four buttons numbered 1 to 4 in each of the regions. The subject was given a sequence of simple instructions and was asked to execute correctly as many of them as possible. The instructions were simple and abstract, for example, *set B to 2, set J to 4* (in the following abbreviated as *B2, J4*). The subject then had to click with the mouse to button 2 in region B and to button 4 in region J. The probability that the subject mismatches instructions was determined for sequences of different lengths. For example, if the subject was instructed to set *B2 J4 V1 X3*, the probability that one or more mismatches occurred was higher than if he was first instructed to set *B2*, then *J4*, then *V1*, and eventually *X3*. Moreover, the error probability was in general higher if the subject was distracted by the secondary task than if he was able focus on the primary task without distraction. Qualitatively, these results are not surprising. But the precise quantitative results—the probabilities determined this way—can be used to ground the decision-theoretic planning approach for the experimental setting on a proper empirical base.

For the rest of the chapter, we consider the problem of planning the optimal strategy for giving a subject two instructions of the kind that we described above. We will refer to this problem as



Stepwise:
System: Set X to 3.
Subject: ... [ok]
System: Set M to 1.
Subject: ... [ok]
System: Set V to 4.
Subject: ... [ok]

Bundled:
System:
Set X to 3,
set M to 1,
set V to 4.
Subject: ... ... [ok]

***Figure 3.2**: The abstract user interface of the instructions experiment.*
(The subjects executed spoken instructions of the kind "set B to 2" by clicking with the mouse to button 2 in region B. Several presentation strategies were tested: stepwise presentation, and the presentation in bundles of 2, 3, and 4 instructions. After executing the given instruction(s), the subject had to confirm completion by clicking on the okay button on the bottom of the screen.)

the *simple instruction problem* (and as the *instruction problem* for sequences with more than two instructions, respectively).

3.1.2 Elements of MDPs

The basic elements of a Markov decision process are states and actions. A state is defined by a set of state features. Actions cause a transition of the system in consideration from its current state to a successor state. In the following sections, we illustrate the basic elements of MDPs with two concrete instances of the simple instruction problem.

3.1.2.1 States and State Features

A *state* describes all aspects of the world relevant for the process to be planned at a particular point in time. The choice of an appropriate set of state features is an important part of the modeling of the problem design. We define each state for the simple instruction problem by the following four state features: (1) N TO GIVE, the number of instructions that remain to be given by the system, (2) N IN BUNDLE, the current size of the instruction bundle, (3) N IN WM, the number of instructions currently in the user's working memory, and (4) CORRECT?, the information if a mismatch has occurred in at least one of the executed instructions so far (here, + means no mismatch, – means at least one mismatch has occurred). The general structure of the MDPs for instances of the instruction problem is summarized in Figure 3.3

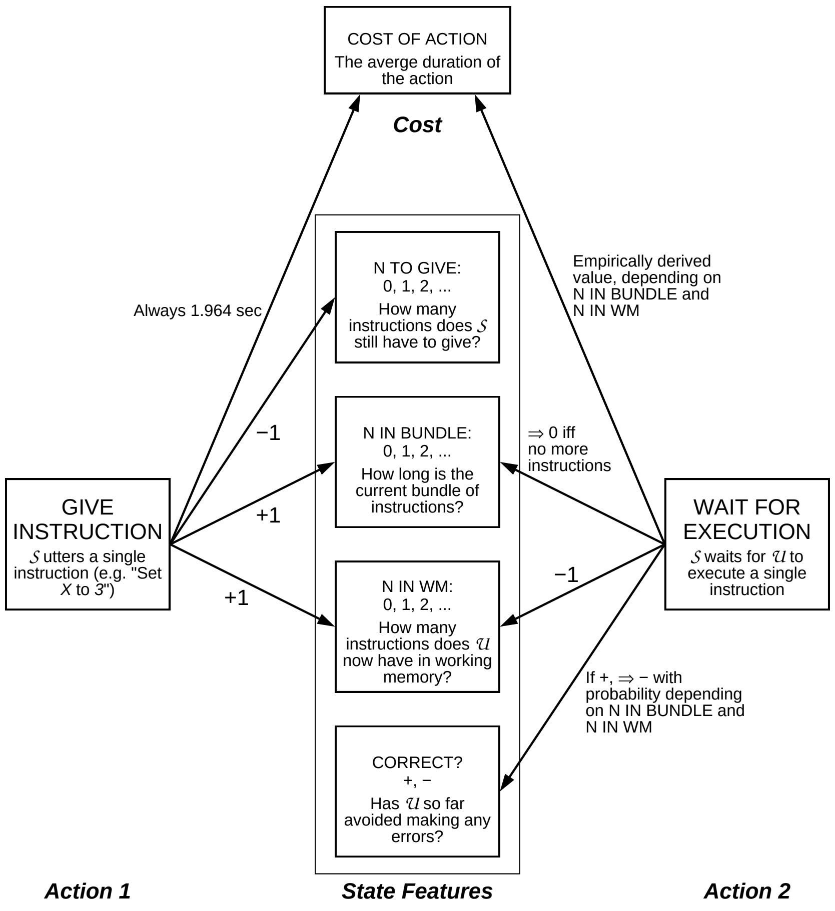

COST OF ACTION
The averge duration of the action
Cost
N TO GIVE: 0, 1, 2, ... How many instructions does $\mathcal{S}$ still have to give?
N IN BUNDLE: 0, 1, 2, ... How long is the current bundle of instructions?
N IN WM: 0, 1, 2, ... How many instructions does $\mathcal{U}$ now have in working memory?
CORRECT? +, − Has $\mathcal{U}$ so far avoided making any errors?
GIVE INSTRUCTION
$\mathcal{S}$ utters a single instruction (e.g. "Set X to 3")
WAIT FOR EXECUTION
$\mathcal{S}$ waits for $\mathcal{U}$ to execute a single instruction
Always 1.964 sec
−1
+1
+1
Empirically derived value, depending on N IN BUNDLE and N IN WM
⇒ 0 iff no more instructions
−1
If +, ⇒ − with probability depending on N IN BUNDLE and N IN WM
Action 1
State Features
Action 2

***Figure 3.3**: General structure of the MDPs for instances of the instruction problem.*
(The actions GIVE INSTRUCTION *and* WAIT FOR EXECUTION *change the world state. Action* GIVE INSTRUCTION *has an effect on the state features* N TO GIVE*,* N IN BUNDLE*, and* N IN WM*;* WAIT FOR EXECUTION *has an effect on* N IN BUNDLE*,* N IN WM*, and* CORRECT?*. Both actions are associated with costs that correspond to their duration.)*

State features are in principle not restricted to simple numeric or symbolic data types. For example, instead of keeping track only of the number of instructions in the user's working memory, we could also consider a list with more specific information of the user's working memory content (cf. Bohnenberger, 2000). There often exist various possibilities to represent the same information about a world state. For example, a list can be encoded by several simpler state features for each element in the list. Some tools for solving MDPs allow only boolean state features, for example, earlier versions of SPUDD[5] (Hoey, St-Aubin, Hu, & Boutilier, 1999), which we will study in comparison with our own implementation in Section 3.1.5.1.

States can be represented *extensionally* (i.e., each state is explicitly named) or *intensionally* (i.e., each state is described by a set of variables: the *state features*). Figure 3.4 indicates the extensional representation by the state denotations $s_0, \ldots, s_{10}$ and the intensional representation by the four state features of each state. Intensional representations are often used in AI planning

[5]SPUDD stands for **S**tochastic **P**lanning **U**sing **D**ecision **D**iagrams.

systems. For example, the world representation used by the STRIPS planner is an intensional representation. An intensional state representation can easily be converted to an extensional one (but not vice versa). Both, the use of the extensional and the intensional representation have certain advantages and disadvantages for the implementation of an MDP and the algorithm to solve it. We will explain our decision to use an extensional representation that is derived from an intensional one in Section 3.1.5.2.

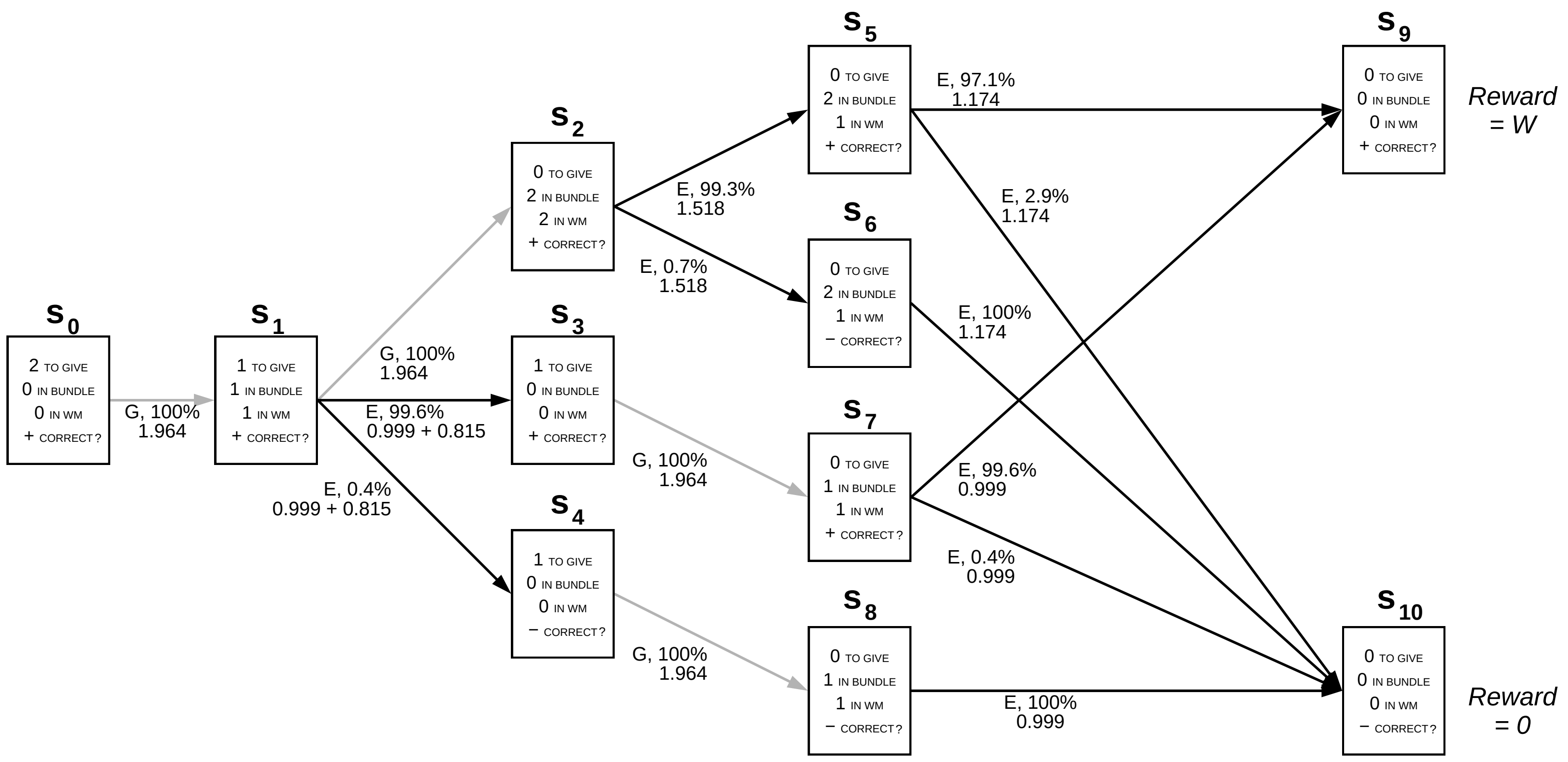
3.1.2.2 Actions and Transitions

Transitions between states correspond to *actions*. In the context of user-adaptive systems, both the user and the system can cause a state change by executing actions. In the simple instruction problem, we consider two actions: (a) The system can give an instruction and (b) the user can execute an instruction (i.e., the system waits for the user to execute the given instruction—this view describes the user's action from the perspective of the system). Figure 3.3 describes the effect of each of the two actions on each of the state features. If an action is deterministic, then it corresponds exactly to one transition in an MDP. Applying a deterministic action in a particular state leads to one particular successor state with probability 1.0. In the simple instruction problem, we consider the system's actions (giving instructions) to be deterministic. As MDPs allow for uncertain outcomes of an action, one action can also correspond to several transitions. Applying an uncertain action in a particular state means that there exist several transitions corresponding to this action, which lead from the current state to different successor states. In the simple instruction problem, we consider the user's actions (executing instructions) to be uncertain. We denote these two actions from the system's perspective as GIVE INSTRUCTION and WAIT FOR EXECUTION (or G and E in Figure 3.4, respectively). Figure 3.4 shows the *transition diagram* of an instance[6] of the simple instruction problem by which the system can plan the optimal strategy to give two instructions. The MDPs constructed according to the generation recipe of Figure 3.3 do not exhibit *cycles*, but MDPs do not need to be acyclic in general.

Each of the transitions in an MDP is annotated with a *probability*—the probability that the world is in the corresponding successor state after the uncertain action is executed. In the simple instruction problem, the user executes an instruction correctly with probability p and makes a mistake with probability $1 - p$. As we have said, there can be more than two transitions for one action. The probabilities of all transitions that correspond to the same action sum up to 1. Tables 3.1 and 3.2 show the transition matrices of the actions GIVE INSTRUCTION and WAIT FOR EXECUTION, respectively. Each matrix is a complete description of a particular action's effects in any possible state. An action that is in practice not applicable in a particular state is assigned a probability of 1.0 for the successor state being the same as the current state (i.e., the action does not change the current state, e.g., applying action GIVE INSTRUCTION in state s_2 does not change the state; see Table 3.1).

Note the following peculiarity about the structure of the MDP of Figure 3.4: Although the desired goal state s_9 cannot be reached anymore once the first instructed has been executed incorrectly, the subject has to execute both instructions in any case. Various assumptions about the error handling can be implemented, for example, that the system stops the interaction immediately after the first incorrect execution of an instructions, or that the system repeats an incorrectly executed instruction once or several times (cf. Section 4.1.4.1).

[6]We speak of an instance of the (simple) instruction problem to denote the initialization of the MDP with a particular reward for the desired goal state.



s0: 2 TO GIVE, 0 IN BUNDLE, 0 IN WM, + CORRECT?
s1: 1 TO GIVE, 1 IN BUNDLE, 1 IN WM, + CORRECT?
s2: 0 TO GIVE, 2 IN BUNDLE, 2 IN WM, + CORRECT?
s3: 1 TO GIVE, 0 IN BUNDLE, 0 IN WM, + CORRECT?
s4: 1 TO GIVE, 0 IN BUNDLE, 0 IN WM, − CORRECT?
s5: 0 TO GIVE, 2 IN BUNDLE, 1 IN WM, + CORRECT?
s6: 0 TO GIVE, 2 IN BUNDLE, 1 IN WM, − CORRECT?
s7: 0 TO GIVE, 1 IN BUNDLE, 1 IN WM, + CORRECT?
s8: 0 TO GIVE, 1 IN BUNDLE, 1 IN WM, − CORRECT?
s9: 0 TO GIVE, 0 IN BUNDLE, 0 IN WM, + CORRECT?
s10: 0 TO GIVE, 0 IN BUNDLE, 0 IN WM, − CORRECT?
G, 100% 1.964
G, 100% 1.964
E, 99.6% 0.999 + 0.815
E, 0.4% 0.999 + 0.815
E, 99.3% 1.518
E, 0.7% 1.518
G, 100% 1.964
G, 100% 1.964
E, 97.1% 1.174
E, 2.9% 1.174
E, 100% 1.174
E, 99.6% 0.999
E, 0.4% 0.999
E, 100% 0.999
Reward = W
Reward = 0

***Figure 3.4**: Transition diagram for the simple instruction problem (2-step sequences).*
(Transitions from the start state to the goal states. G *denotes the action* GIVE INSTRUCTION *and* E *corresponds to the action* WAIT FOR EXECUTION*. The features listed for the states are explained in Figure 3.3. The transitions are annotated with the costs and probabilities empirically determined in the experiment described in Section 3.1.1. The states* $\mathbf{s_9}$ *and* $\mathbf{s_{10}}$ *are goal states with different rewards.)*

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
s_0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
s_1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
s_2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
s_3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
s_4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
s_5	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
s_6	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
s_7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
s_8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
s_9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
s_{10}	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

***Table 3.1**: Transition matrix for the action* GIVE INSTRUCTION.

(Transition matrix for the action GIVE INSTRUCTION *under the condition that the user is distracted by the secondary task described in the text. All transitions are deterministic.)*

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
s_0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
s_1	0.0	0.0	0.0	0.996	0.004	0.0	0.0	0.0	0.0	0.0	0.0
s_2	0.0	0.0	0.0	0.0	0.0	0.993	0.007	0.0	0.0	0.0	0.0
s_3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
s_4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
s_5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.971	0.029
s_6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
s_7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.996	0.004
s_8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
s_9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
s_{10}	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

***Table 3.2**: Transition matrix for the action* WAIT FOR EXECUTION.

(Transition matrix for the action WAIT FOR EXECUTION *under the condition that the user is distracted by the secondary task described in the text. Trying to execute instructions in state s_0, s_3, s_4, s_9, and s_{10} although* N IN WM $= 0$ *does not change the state).*

3.1.2.3 Costs and Rewards

Each action has a *cost*, each goal state is associated with a *reward*. Decision-theoretic planning trades off the costs of a sequence of actions with the expected reward for reaching a goal. The cost function, assigning a cost to each action executed in a particular state, must have the same range as the reward function, assigning a reward to each state. The interpretation of an action's cost allows for some flexibility, but it has to comply with the interpretation of the reward(s) for goal state(s).

In the simple instruction problem, we associate the cost of an action with the time that it needs to be carried out. For example, giving an instruction might take x sec, executing an instruction might take y sec. As we define x and y to be the costs of the corresponding actions in terms of the actions' duration, we must consider the reward for the preferred goal state in terms of time, too. For example, we can say that executing both instructions without making a mistake is worth as much as z sec.

Cost functions can be represented in matrices analogous to transition functions (cf. the transition matrices of Table 3.1 and 3.2). Having introduced all elements of a Markov decision process, we can now define a Markov decision process as follows.

Definition 3.1 *A (fully observable) Markov decision process is a tuple* $< \mathcal{S}, \mathcal{A}, T, C, R >$*, where*

- $\mathcal{S} = \{s_1, \ldots, s_N\}$ *is a finite set of world states*
- $\mathcal{A} = \{a_1, \ldots, a_K\}$ *is a finite set of actions*
- $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ *is the* state-transition function, *specifying for each pair of world state and action a probability distribution over world states. We write* $T(s, a, s')$ *for the probability of ending in state* s'*, given that action* a *is executed in state* s
- $C : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ *is the* cost function, *specifying the cost for applying a particular action in a particular state. We write* $C(s, a)$ *for the cost of taking action a in state s; and*
- $R : \mathcal{S} \rightarrow \mathbb{R}$ *is the* reward function, *specifying the expected immediate reward gained for being in a particular state. We write* $R(s)$ *for the expected reward*[7] *for being in state s.*

The solution of a planning problem is a sequence of actions that achieves a desired state. In the traditional planning paradigm, goal states are defined by logical expression, and a plan consists of a sequence of operators that are applied until the initial world state is transformed into the goal state (cf. Section 2.2.2). In the decision-theoretic planning paradigm, a policy maps each state to an action, and this action is determined by its expected utility (compared to the expected utility of alternative actions).

Definition 3.2 *A* policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ *is a mapping from the finite set of world states* $\mathcal{S}$ *to the finite set of actions* $\mathcal{A}$*.*

The expected utility of an action is determined by the values of the states that can be reached by this action. Therefore, the *goal states* are associated with a numeric reward. In the simple instruction problem, the goal states are those in which no instructions remain to be given by the system and no instructions in the subject's memory remain to be executed. In the example of Figure 3.4, there are two states for which these conditions hold. We give a positive reward only to the one in which no mismatch has occurred during the execution of the instructions (i.e., CORRECT? = *yes*). Applying the iterative solution algorithm that we will describe in Section 3.1.4 means to propagate these rewards from the goal states via the probabilistic transitions through the complete MDP, thereby allowing to compute the *expected utility* of every single state in the process.

3.1.3 Evolution of an MDP

A discrete-time stochastic dynamical system consists of a state space and probability distributions describing possible state transitions. We assume that the modeled process evolves in *stages*: The transition from stage t to the next stage $t+1$ corresponds to any of the actions (or events) specified

[7]Some authors (e.g., Boutilier et al., 1999) prefer a reward function of the kind $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. This allows to use the reward function to express the cost of an action in terms of a negative reward and thereby integrates cost and reward function. We assume that merely being in a particular state is already rewarded, no matter if the decision maker performs any action in that state or not, and consider the cost function separately.

in the MDP. We refer to the set of all stages as $\mathcal{T}$. The progression through stages is only roughly analogous to the passage of time. In the basic MDP framework, each execution of an action corresponds to proceeding to the next stage in the process. As usually not all actions take the same amount of time, it is not clear how the notion of time can be captured in MDPs. Two different roles of time in the context of MDPs were sketched in Section 1.4.2. We will describe the formal consequences of considering time as a pure cost factor and time restricted by a fixed deadline in Chapter 4.

We can model uncertainty about the state s of a process at some stage t as a random variable S^t that takes values from $\mathcal{S}$. The assumption of *forward causality* requires that the variable S^t does not depend directly on the value of future variables $S^k, k > t$. In analogy, we can model uncertainty about the action a taken in a process at some stage t as a random variable A^t. The *history*[8] h of a process up to the stage t can then be defined by the sequence of pairs

$$< S^0, A^0 >, < S^1, A^1 >, \ldots, < S^{t-1}, A^{t-1} >.$$

According to the *Markov property,* it is commonly assumed that a state contains all information necessary to predict its successor state(s). That is, any information about the history of the considered process relevant to predict the next state is captured in the state itself. Formally, the Markov property holds if

$$P(S^{t+1} \mid S^t, S^{t-1}, \ldots, S^0) = P(S^{t+1} \mid S^t). \tag{3.1}$$

The Markov assumption claims that the knowledge of the present state includes all relevant information about the past to make predictions about the future. Luenberger (1979) shows that any non-Markovian model can be converted to an equivalent though larger Markov model.

Moreover, it is sometimes assumed that the effects of an action depend only on the current state and not on the stage at which an action is executed. If the probability distribution predicting the next state is the same regardless of stage, the model is said to be *stationary*.

The number of stages considered in a decision-theoretic planning process is called the *planning horizon*. We distinguish *finite-horizon* and *infinite-horizon* problems. In *finite-horizon problems,* the performance of a system interacting with its user is evaluated over a fixed, finite number of stages. The aim is to maximize the total expected reward associated with a course of interaction of a fixed length. *Infinite-horizon problems* appear in the context of user-adaptive systems in the sense of the horizon of the interaction being unknown: The interaction of the system with its user will inevitably come to an end at some point. This holds also for the problems that we consider in this thesis. The total reward gained in an infinite-horizon problem can in general be unbounded, which means that any policy could be arbitrarily good or bad if it was executed for long enough. Therefore, a *discount factor* is introduced, ensuring that rewards and costs at earlier stages count more than those at later stages.

3.1.4 Solving MDPs

The problem of solving a finite-horizon MDP can be rephrased as finding a strategy that maximizes the total expected reward over some fixed, finite number of stages t. Such a strategy is commonly called a *policy* in decision-theoretic planning. Two types of policies are distinguished: stationary and non-stationary policies. A *stationary policy* π is defined as a mapping from states to actions: $\pi : \mathcal{S} \rightarrow \mathcal{A}$, that is, a stationary policy describes which action to take in a particular state, but

[8]A history is also called *trajectory*.

regardless of stage. A non-stationary policy additionally considers the stage of the process to specify the action to be taken. Let $\mathcal{T}$ be the set of all stages. A *non-stationary policy* π_t is a mapping from states and stages to actions: $\pi_t : \mathcal{S} \times \mathcal{T} \to \mathcal{A}$, that is, a non-stationary policy describes which action to take in a particular state and at a particular stage in the process. Non-stationary policies will be particularly important in the scenarios that require to consider time restricted by a fixed deadlines (see Section 4.4).

3.1.4.1 Finite-Horizon Problems

To describe the value of a state at a particular stage, we define a *value function* $V : \mathcal{S} \times \mathcal{T} \to \mathbb{R}$ to be a mapping from states and stages to reals. Two assumptions about value functions are commonly made in the MDP literature: The value function is *time-separable* and *additive*.

Boutilier et al. (1999) describe time-separability and additivity as follows: A value function is *time-separable* if it is a stage-independent combination of the reward and the cost function. That means, the costs and rewards can depend on the stage, but the function that combines them must be independent of the stage. For the problems that we consider in this thesis, neither the function combining costs and rewards nor the cost and reward functions themselves depend on the stage. A value function is *additive* if the combination function is a sum of the rewards and costs accumulated over the history's stages.

Under these assumptions, Bellman (1957) shows that the expected value of a policy at any state can be computed using the *t-stage-to-go value function* $V_t{}^\pi$ with $V_0{}^\pi = R(s)$ and

$$V_t^\pi(s) = R(s) + C(\pi(s,t)) + \sum_{s' \in \mathcal{S}} P(s' \mid \pi(s,t), s) V_{t-1}^\pi(s'). \tag{3.2}$$

The recursive structure of the formula relating the value of a state to the immediate expected rewards for being in a state already indicates the iterative notion that characterizes the dynamic programming solution algorithms.

A policy is *optimal* if $V_t^\pi(s) \geq V_t^{\pi'}(s)$ for all policies π' and all $s \in \mathcal{S}$. The optimal t-stage-to-go value function V_t^* is defined as the value function of any optimal t-horizon policy. Bellman's *principle of optimality* forms the basis of the stochastic dynamic programming algorithms used to solve MDPs. It describes the relationship between the optimal value function at stage t and the optimal value function at stage $t-1$:

$$V_t^*(s) = R(s) + \max_{a \in \mathcal{A}} \{C(a) + \sum_{s' \in \mathcal{S}} P(s' \mid a, s) V_{t-1}^*(s')\}. \tag{3.3}$$

The optimal finite-horizon policy at stage t is described by the formula

$$\pi_t^*(s) = \arg\max_{a \in \mathcal{A}} \{R(s) + \sum_{s' \in \mathcal{S}} P(s' \mid a, s) V_{t-1}^*(s')\}. \tag{3.4}$$

Value Iteration Equation 3.3 forms the basis of a standard dynamic programming approach for solving finite-horizon Markov decision problems called *value iteration*. Value iteration takes the immediate expected rewards as the initial value for being in a state, that is, $V_0^* = R$ for all $s \in \mathcal{S}$, and then iteratively applies Equation 3.3 to compute the expected values of all states and at all stages up to the horizon t.

We can now apply value iteration to our simple instruction problem. We first assume a reward function that assigns a reward of 2.0 for being in state s_9 and no reward for being in any of the other states. Here are two examples for the computation of the expected utilities of executing an action in a particular state and at a particular stage[9]:

$$\begin{aligned} EU_1(s_5, \mathsf{E}) &= 0.971\ V_0^*(s_9) + 0.029\ V_0^*(s_{10}) - 1.174 \\ &= 0.768 = V_1^*(s_5) \end{aligned}$$

$$\begin{aligned} EU_2(s_2, \mathsf{E}) &= 0.993\ V_1^*(s_5) + 0.007\ V_1^*(s_6) - 1.518 \\ &= -0.764 = V_2^*(s_2) \end{aligned}$$

The expected utility of waiting for the user to execute an instruction in state s_5 with one stage to go is the weighted sum of the immediate rewards for being in state s_9 or s_{10}, minus the cost of action E, which reflects the average time that the user needs to execute an instruction in that situation. The expected utility of waiting for the user to execute an instruction in state s_2 with two stages to go is the weighted sum of the values for being in state s_5 or s_6, minus the cost of action E. In the states s_5 and s_2, there exists no alternative action. Therefore, $V_1^*(s_5)$ coincides with $EU_1(s_5, \mathsf{E})$ and $V_2^*(s_2)$ coincides with $EU_2(s_2, \mathsf{E})$. If we compute the value for being in state s_1 with three stages to go, we have to maximize over the expected utilities of the two actions G and E:

$$\begin{aligned} EU_3(s_1, \mathsf{G}) &= 1.0\ V_2^*(s_2) - 1.964 \\ &= -2.728 \end{aligned}$$

$$\begin{aligned} EU_3(s_1, \mathsf{E}) &= 0.996\ V_2^*(s_3) + 0.004\ V_2^*(s_4) - 0.999 - 0.815 \\ &= -2.793 \end{aligned}$$

$$\begin{aligned} V_3^*(s_1) &= R(s_1) + \max\{EU_3(s_1, \mathsf{G}), EU_3(s_1, \mathsf{E})\} \\ &= 0.0 + \max\{-2.728, -2.793\} = -2.728 \end{aligned}$$

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
$V_0{}^*$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
$V_1{}^*$	-1.964	-1.814	-1.518	-1.964	-1.964	0.768	-1.174	0.993	-0.999	2.0	0.0
$V_2{}^*$	-3.778	-3.482	-0.764	-0.971	-2.963	0.768	-1.174	0.993	-0.999	2.0	0.0
$V_3{}^*$	-5.446	-2.728	-0.764	-0.971	-2.963	0.768	-1.174	0.993	-0.999	2.0	0.0
$V_4{}^*$	-4.692	-2.728	-0.764	-0.971	-2.963	0.768	-1.174	0.993	-0.999	2.0	0.0

***Table 3.3**: The optimal n-stage-to-go ($n = 0, \ldots, 4$) value functions for the simple instruction problem with a reward of 2.0 for being in state $\mathbf{s_9}$.*

Four iterations of the value iteration algorithm result in the value functions up to stage 4 presented in Table 3.3. The value of states close to the goal states s_9 and s_{10} do not change anymore after the first iteration. The state s_1 needs three iterations, the state s_0 four iterations until its value does not change anymore. $V_4^* = V_n^*$ for all $n \geq 5$.

[9]The numeric results in this section are all rounded in the third position after decimal point.

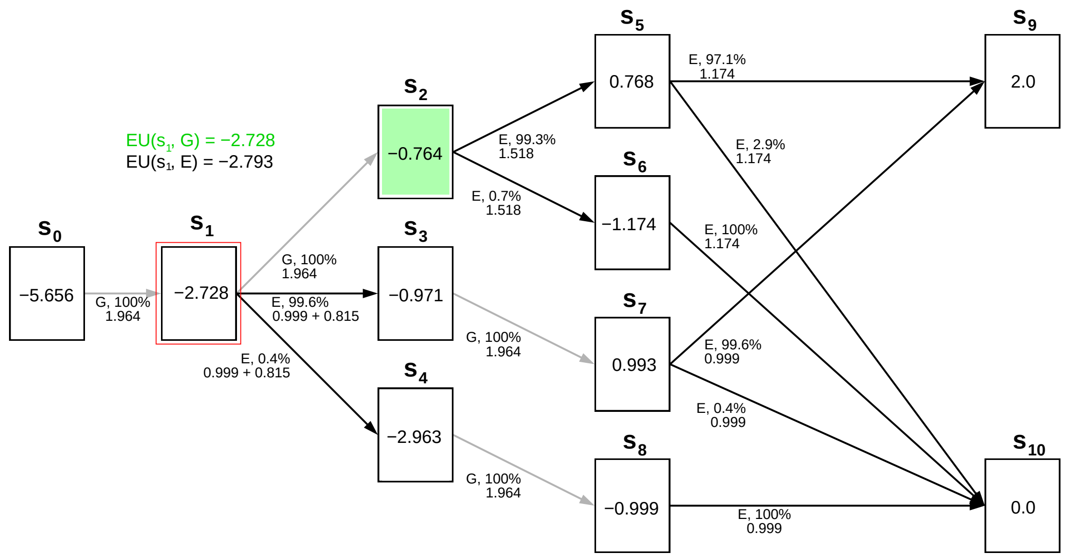


***Figure 3.5**: Simple instruction problem: State utilities after 4 iterations; reward = 2.0.*
(Value iteration converges after 4 iterations, resulting in the given state utilities. Given the rather low reward for executing all instructions correctly, the system decides to bundle the two instructions. That is, it selects action G *in state* $\mathbf{s_1}$*. The highlighted state is the successor state after the decision made in state* $\mathbf{s_1}$*.)*

Figure 3.5 relates the optimal 4-stage-to-go value function to the transition diagram. The only state in which the system can make a decision is state s_1. Assuming a reward of 2.0 for state s_9 results in the expected utilities $EU_4(s_1, \mathsf{G}) = -2.728$ and $EU_4(s_1, \mathsf{E}) = -2.793$. Therefore, the system's rational decision in this state is to choose action G, that is, to give the second instruction and thereby bundle two instructions.

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
$V_0{}^*$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0
$V_1{}^*$	-1.964	-1.814	-1.518	-1.964	-1.964	8.536	-1.174	8.961	-0.999	10.0	0.0
$V_2{}^*$	-3.778	-3.482	6.950	6.997	-2.963	8.536	-1.174	8.961	-0.999	10.0	0.0
$V_3{}^*$	-5.446	5.143	6.950	6.997	-2.963	8.536	-1.174	8.961	-0.999	10.0	0.0
$V_4{}^*$	3.179	5.143	6.950	6.997	-2.963	8.536	-1.174	8.961	-0.999	10.0	0.0

***Table 3.4**: The optimal n-stage-to-go (*$n = 0, \ldots, 4$*) value functions for the simple instruction problem with a reward of 10.0 for being in state* $\mathbf{s_9}$*.*

If we assume a higher reward for getting to state s_9, say $R(s_9) = 10.0$, then the system's decision in state s_1 is different. The expected utilities are $EU_4(s_1, \mathsf{G}) = 4.986$ and $EU_4(s_1, \mathsf{E}) = 5.143$. Therefore, the system's rational decision under these circumstances is to choose action E, that is, to wait for the user executing the given instruction and thereby give the two instructions of the complete sequence stepwise. Table 3.4 shows the evolution of the optimal value function up to stage 4, assuming a reward of 10.0 for being in state s_9. Figure 3.6 relates the optimal 4-stage-to-go value function to the transition diagram.

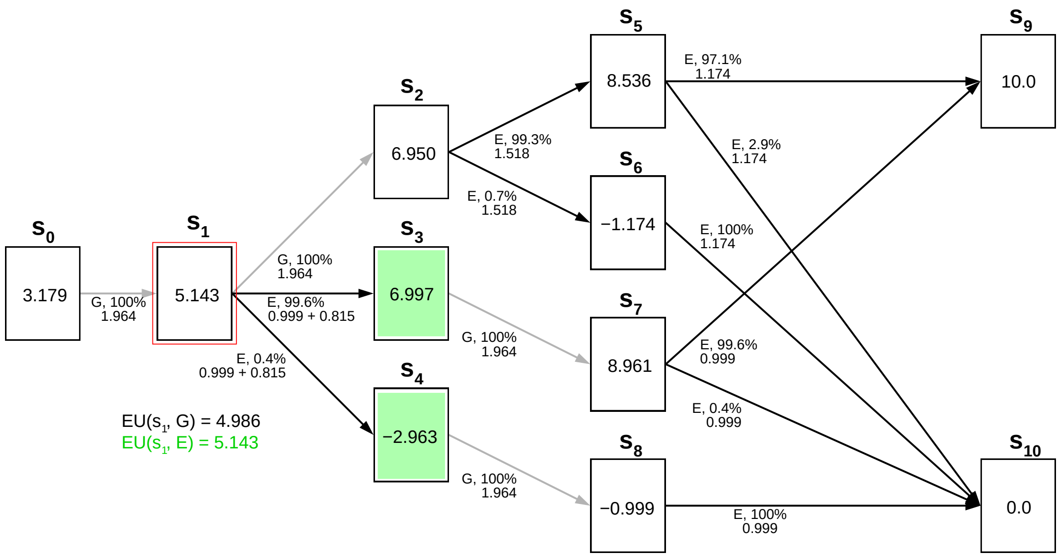


***Figure 3.6**: Simple instruction problem: State utilities after 4 iterations; reward = 10.0.*
(Value iteration converges after 4 iterations, resulting in the given state utilities. Given the rather high reward for executing all instructions correctly, the system decides to give the two instructions stepwise. That is, it selects action E *in state* $\mathbf{s_1}$*. The highlighted states are the possible successor states after the decision made in state* $\mathbf{s_1}$*.)*

3.1.4.2 Infinite-Horizon Problems

In some cases, there is no restriction about how many stages exactly there are to go in an interaction of a system with its user. Instead, the system might want to ensure that the user reaches desired states as soon as possible. Such problems can be considered as infinite-horizon discounted Markov decision problems. The basic idea is to compute a stationary policy, that is, a policy that recommends the same action at each stage in a continuous and (in principle) infinite process. Thinking of a process for which the horizon is just not known might yield a better intuition for the concept of an infinite-horizon problem. The formula satisfied by the optimal policy for an infinite-horizon problem differs from the one for a finite-horizon problem (Equation 3.3) only in the *discount factor* $\gamma \in [0, 1)$. It is used to give more emphasis to earlier rewards than to rewards gained at later stages in the process:

$$V^*(s) = R(s) + \max_{a \in \mathcal{A}} \{C(a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid a, s) V^*(s')\} \tag{3.5}$$

The optimal infinite-horizon policy is described by the formula

$$\pi_t^*(s) = \arg\max_{a \in \mathcal{A}} \{R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid a, s) V_{t-1}^*(s')\} \tag{3.6}$$

Howard (1960) shows that there always exists an optimal stationary policy for infinite-horizon problems. It can be computed by a slightly modified version of the value iteration algorithm, which takes the discount factor into consideration:

$$V_{t+1}(s) = R(s) + \max_{a \in \mathcal{A}} \{C(a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid a, s) V_t(s')\} \tag{3.7}$$

Starting with arbitrary assignments of values for $V_0(s)$, the sequence of function V_t converges linearly to the optimal value function $V^*(s)$ after some finite number of iterations n. The number of iterations n depends on a *stopping criterion* that generally involves measuring the difference between V_{t+1} and V_t. The iteration process stops if this difference is sufficiently small. Puterman (1994) discusses different stopping criteria and the convergence of the value iteration algorithm for infinite-horizon problems.

Policy Iteration Howard (1960) introduces an alternative algorithm to value iteration for infinite-horizon problems. The approach is called *policy iteration*. While value iteration iteratively improves the estimated value function, policy iteration modifies the policy directly. The key idea of policy iteration is to start from a random policy π_0 and then iteratively improve the policy by computing π_{i+1} from π_i. The policy iteration algorithm alternates two steps, called *policy evaluation* and *policy improvement*, until convergence:

- **Policy evaluation:** For each $s \in S$ compute the value function $V^{\pi_i}(s)$ based on the current policy π_i. $V^{\pi_i}(s)$ is the value of s if $\pi_i(s)$ was executed as the next action.
- **Policy improvement:** For each $s \in S$, find the action a^* that maximizes

$$Q_{i+1}(a, s) = R(s) + C(a) + \gamma \sum_{s' \in S} P(s' \mid a, s) V^{\pi_i}(s') \tag{3.8}$$

  and update $\pi_{i+1} = a^*$ if $Q_{i+1}(a^*, s) > V^{\pi_i}(s)$; otherwise $\pi_{i+1}(s) = \pi_i(s)$.

The algorithm terminates when $\pi_{i+1}(s) = \pi_i(s)$ for all $s \in S$, that is, when the policy improvement step yields no further change in the expected utilities. The policy improvement step is guaranteed to improve the policy (if it is not already optimal).

For infinite-horizon problems, both algorithms, value iteration and policy iteration, result in the optimal policy (see, e.g., Puterman, 1994). Puterman (1994) shows also that policy iteration usually requires fewer steps in total than value iteration (or at least not more) because it makes policy improvements based on more accurate information. Boutilier et al. (1999) emphasize that in practice, policy iteration tends to converge in a lot less iterations than does value iteration. Koller and Parr (2000) propose an approximative version of policy iteration for factored MDPs. There exist extensions of the policy iteration algorithm, such as *modified policy iteration* (Puterman & Shin, 1978) and *asynchronous version of policy iteration* (Bertsekas, 1987), that aim at further speeding up the convergence of the algorithm.

We do not consider policy iteration in more detail because it is not applicable to finite-horizon problems and therefore cannot be used for many of the problems that we consider in the context of user-adaptive systems. Moreover, as we will explain in Section 3.1.7, the crucial problem with MDPs is not the performance of the algorithms, but rather the exponential growth of the state space.

3.1.5 State Space Representations

We have distinguished extensional and intensional representations of the state space in Section 3.1.2. This distinction is used to describe a state space representation on a conceptual level. In this section, we will compare two approaches to solve MDPs. The first approach, SPUDD, is based on an extensional representation and provides a general framework to solve MDPs. SPUDD can be considered as a state of the art AI approach for solving MDPs. It follows the tradition of AI planning to represent a state by the relevant attributes that describe the state. Other approaches (e.g., Feng & Hansen, 2002) use SPUDD as a basis and try to improve SPUDD's performance with special techniques. The second approach is based on an intensional representation and provides problem-tailored solutions. The key idea of this approach is the object-oriented representation of state-transition diagrams. On the one hand, the approach sticks with the traditional OR approach in that it explicitly enumerates states. On the other hand, it goes beyond the traditional OR approach in that it emphasizes the object-oriented programming paradigm for the efficient representation of an MDP and the convenient incorporation of a solution algorithm like value or policy iteration. Both approaches have advantages and disadvantages.

3.1.5.1 SPUDD

Hoey et al. (1999) introduce SPUDD[10], a method that applies a version of the policy iteration algorithm (called *structured policy iteration, SPI*) on a *factored*[11] *representation* of an MDP called *algebraic decision diagrams (ADDs)*. ADDs allow to represent the dynamics of a process (in terms of the action specifications), the value function, and the policy in a compact *decision graph*. The SPI algorithm derives from an earlier approach that employs decision trees to represent value function and policy (see, e.g., Boutilier, Dearden, & Goldszmidt, 1995). The ADD formalism requires an MDP state space to be represented by a set of boolean variables $\mathcal{X} = X_1, \ldots, X_n$. The fact that ADDs require boolean variables is not a general restriction (any n-ary variable can be expressed by several boolean variables), but it is the reason that versions of SPUDD earlier than SPUDD 2.0 allow only boolean variables for the specification of the process dynamics. The system is perpetually improved[12]. For example, it is meanwhile possible to use multi-valued variables for the specification of our illustrating example, the simple instruction problem. However, SPUDD internally translates the multi-valued specification into ADDs, such that no performance increase compared to earlier version of the system results from this improvement.

The dynamics of an MDP can be captured in terms of a dynamic Bayesian network (see Section 2.1.3.4) with two time slices for each action $a \in \mathcal{A}$. The links represent causal dependencies between variables of $\mathcal{X}$ and $\mathcal{X}'$ at two consecutive stages. Figure 3.7 shows the DBN representation of the action WAIT FOR EXECUTION.

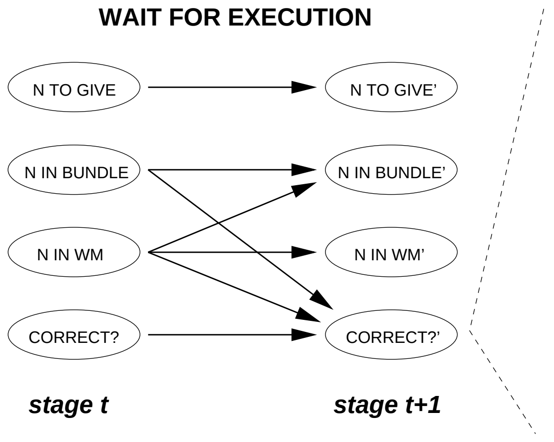
SPUDD uses ADD representations of the conditional probability tables instead of working on the inefficient tabular representation. Figure 3.8 shows the tree structure of the CPT for the variable CORRECT?. It is to be read like "if at stage t CORRECT? equals *yes*, N IN BUNDLE equals *two*, and N IN WM equals *two*, then after applying WAIT FOR EXECUTION the probability that CORRECT? equals *yes* at stage $t + 1$ is 0.993", and so forth. SPUDD internally translates the tree structure to the binary tree structure of an ADD.

SPUDD 3.0 was the first version of SPUDD to allow the specification of action costs within action effects. In the simple instruction problem, the cost of the action WAIT FOR EXECUTION depends on the state features N IN BUNDLE and N IN WM, for example, executing the first instruction of a bundle of two on average takes 1.518sec, while executing the second one of a bundle of two takes on average only 1.174sec. Modeling this distinction in the actions' costs in SPUDD 2.0

[10]SPUDD stands for **S**tochastic **P**lanning **U**sing **D**ecision **D**iagrams.

[11]The term "factored representation" in this context means "feature-based".

[12]At the time of submission of this thesis, the latest version of SPUDD is 3.4.0. The ongoing improvements of SPUDD are documented at `http://www.cs.ubc.ca/spider/staubin/Spudd/`.



N IN BUNDLE	N IN WM	CORRECT?	P(CORRECT?' = yes)
two	two	yes	0.993
one	two	yes	0.0
zero	two	yes	0.0
two	two	no	0.0
one	two	no	0.0
zero	two	no	0.0
two	one	yes	0.971
one	one	yes	0.996
zero	one	yes	0.0
two	one	no	0.0
one	one	no	0.0
zero	one	no	0.0
two	zero	yes	0.0
one	zero	yes	0.0
zero	zero	yes	0.0
two	zero	no	0.0
one	zero	no	0.0
zero	zero	no	0.0

***Figure 3.7**: 2-time-slice DBN representation of the action* WAIT FOR EXECUTION
(Links represent causal dependencies between variables of stage t and stage $t + 1$. The conditional probability table for variable CORRECT?$'$ *reveals the inefficiency of representing causal dependencies in a tabular representation.)*

required to model three different variants of the action WAIT FOR EXECUTION: (1) WAIT FOR EXECUTION OF FIRST OF ONE, (2) WAIT FOR EXECUTION OF FIRST OF TWO, and (3) WAIT FOR EXECUTION OF SECOND OF TWO, each of them with a different cost. This circumstance made the specification of the simple instruction problem in SPUDD representation quite inconvenient. It required the one who specifies the problem to ensure that each variant of the action WAIT FOR EXECUTION is applicable only in the right situation. There is a complete and commented specification of the simple instruction problem for SPUDD 2.0 in Appendix A.

We apply SPUDD to the two instances of the simple instruction problem that we have already considered in Section 3.1.4: once with a reward of 2.0 and once with a reward of 10.0 for the goal state in which all instructions have been given and executed correctly. Figure 3.9 and 3.10 show the resulting policies: As expected, they propose to bundle two instructions in the case of a low reward ($R(s_9) = 2.0$) and to give the two instructions stepwise in the case of a high reward ($R(s_9) = 10.0$).

The strength of SPUDD is its generality. In principle, it is possible to specify arbitrary MDPs such that they can be understood and solved by SPUDD. Hoey et al. (1999) have demonstrated the performance of their approach on a class of MDPs with up to 63 million states. This number of states corresponds to a state space spanned by 25 state features, which is sufficient for certain classes of real world problems. With APRICODD[13], St-Aubin, Hoey, and Boutilier (2001) propose a method for approximative dynamic programming, which they apply to problems with state spaces that comprise up to 34 billion states (which corresponds to 35 state features). APRICODD uses the same representation as SPUDD (algebraic decision diagrams) and generates near-optimal value functions and policies. There exists work by Feng and Hansen (2002) based on the SPUDD approach, that applies symbolic heuristic search on a factored representation of Markov decision processes. The approach aims at further constraining the computational effort to states that are reachable from a starting state. The assumption that a starting state is known would, for example, not be problematic for instances of the instruction problem. In other domains (e.g., giving

[13]APRICODD stands for **APpR**ox**I**mate Policy **CO**nstruction Using **D**ecision **D**iagrams.

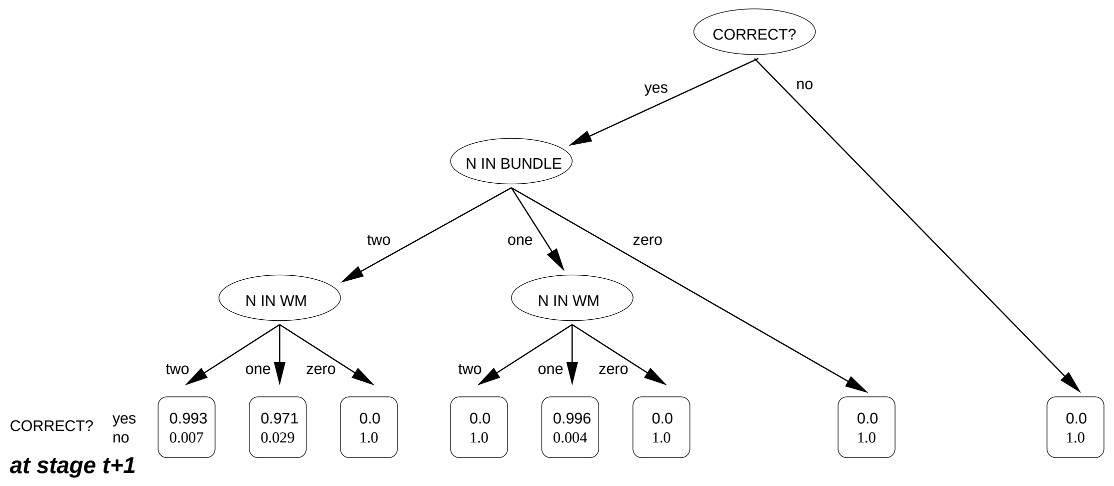


Figure 3.8*: Tree structure of the CPT of the variable* CORRECT?*.*
(The tree structure of the CPT of the variable CORRECT? *of the DBN representation for the action* WAIT FOR EXECUTION *is more compact than the tabular representation. It is to be read top down: For example, after the action* WAIT FOR EXECUTION *at stage t, the value of the variable* CORRECT? *at state $t+1$ is* no *with a probability of* 0.996 *if at stage t* CORRECT? *equals* yes*,* N IN BUNDLE *equals* one*, and* N IN WM *equals* one*. SPUDD internally translates the tree structure into the binary tree structure of an ADD.)*

navigation recommendations in an airport building—cf. Section 4.2) it might be unnecessarily restrictive.

Both methods, SPUDD and APRICODD, are described at full length by Hoey, St.Aubin, Hu, and Boutilier (2000). The authors allude the requirement that variables be boolean as one major drawback. When multi-valued variables are split up into boolean variables, the structure contained in the original problem specification can in general not be preserved completely, which usually results in a state space larger than the original. For domains with relatively few multi-valued variables, SPUDD is assumed not to be handicapped by the requirement of variable splitting.

Apart from this conceptual drawback, our experience with SPUDD has shown that the specification of MDPs in the SPUDD representation is rather inconvenient, even for small problems such as the simple instruction problem. Specifications of MDPs in the SPUDD representation are complex, difficult to maintain, and illegible—even though their structure might in principle be simple to describe (e.g., the description of the simple instruction problem in Figure 3.3). Preserving the complete structure inherent to the problem requires meticulously constraining the applicability of each action. Since we have not done this to the full extend for the simple instruction problem, the resulting policies map optimal actions to some unreachable states. For example, the policy assuming a reward of 10.0 for being in state s_9 (see Figure 3.10) specifies an action for N TO GIVE = one, N IN BUNDLE = two, N IN WM = zero, and CORRECT? = no. This situation cannot occur in practice.

3.1.5.2 Object-Oriented State-Transition Diagrams

The SPUDD approach, which bases the value iteration algorithm on an intensional representation of the planning problem, ties in with the traditional AI paradigm. The main argument in favor of this approach is its ability to capture some, much, or ideally all of the structure inherent to the

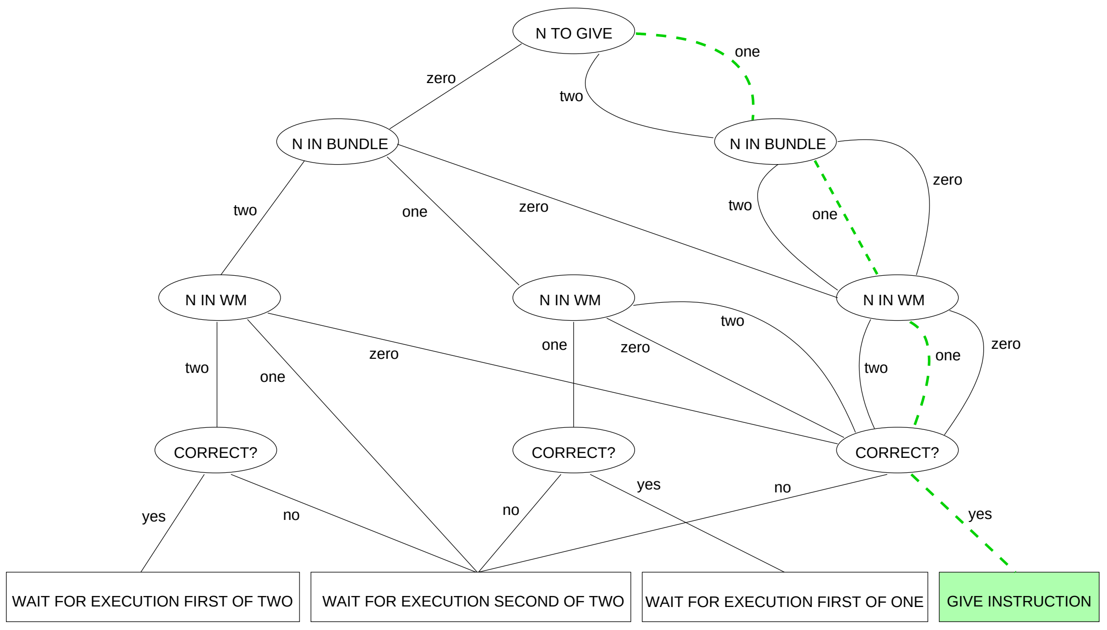


***Figure 3.9**: Policy for the simple instruction problem produced by SPUDD; reward = 2.0. (SPUDD produces a tree structured representation of the policy. It is to be read top down: As an example, we have highlighted (with dashed lines and shading) the action proposed by the policy for the situation in which the system has given the fi rst instruction, the bundle size and the number of instructions in the user's working memory is one, and not mistake has been made in executing instructions so far. The proposed action is to give the next instruction—which is in accordance with the policy represented for the case with low reward in Figure 3.5.)*

process to be planned for. In this respect, decision-theoretic planning based on a factored representation is clearly superior to earlier approaches that base on inefficient tabular representations (matrices) to represent the domain.

However, an alternative that suggests itself with regard to capturing as much as possible of the structure inherent to the process to be planned is to model the structure of the process explicitly. A straightforward approach to do so is offered by *object-oriented programming*. Object-oriented programming provides everything that is necessary to represent a Markov decision process concisely. States are modeled as *objects*, state feature are captured as object *attributes*. Transitions between states are represented as links (references/pointers to other *instances* of the class state node) between objects, and transition probabilities and costs are stored in direct adjacency to the links. During the computation of the optimal policy, values can be propagated through the object-oriented MDP via message passing.

Note that what we introduce in this section is not a new algorithm, but merely the object-oriented implementation of an existing algorithm, which we call *object-oriented value iteration*. We find the way of implementing the approach worth being explained in some detail as it diverges from the implementation suggested by the vast majority of state of the art AI approaches for decision-theoretic planning (e.g., Boutilier et al., 1999; Feng & Hansen, 2002). This applies in particular for the underlying representation used to implement value iteration. To our knowledge, an object-oriented version of value iteration is not explicitly described in decision-theoretic

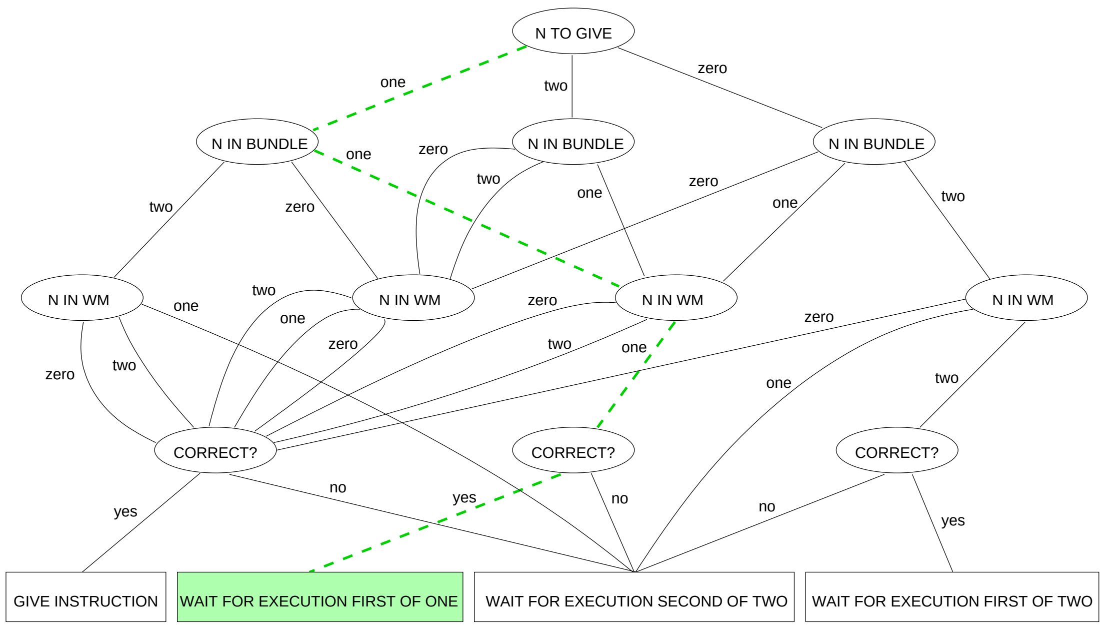

N TO GIVE
N IN BUNDLE
N IN WM
CORRECT?
one
two
zero
yes
no
GIVE INSTRUCTION
WAIT FOR EXECUTION FIRST OF ONE
WAIT FOR EXECUTION SECOND OF TWO
WAIT FOR EXECUTION FIRST OF TWO

***Figure 3.10**: Policy for the simple instruction problem produced by SPUDD; reward = 10.0. (SPUDD produces a tree structured representation of the policy. We have highlighted (with dashed lines and shading) the action proposed by the policy for the situation in which the system has given the first instruction, the bundle size and the number of instructions in the user's working memory is one, and no mistake has been made in executing instructions so far. The proposed action is to wait for the execution of the first instruction—which is in accordance with the policy represented for the case with high reward in Figure 3.6.)*

planning literature. We have used object-oriented value iteration already in the work described by Bohnenberger and Butz (2000), and Bohnenberger (2000). All results presented in Chapters 4, 5, and 7 were produced on the basis of the object-oriented approach.

Figure 3.11 describes the *state node class* with its main attributes and operation. Usually a bunch of other (auxiliary) attributes and operations will be necessary for the actual implementation, which we do not describe here as they are not essential for the modeling. However, the class description in Figure 3.11 contains all information necessary to generate a one-to-one image of the Markov decision process of an simple instruction problem instance.

Two phases can be distinguished in solving an MDP with the object-oriented approach: (1) generating the object-oriented representation of the Markov decision process and (2) applying value iteration (or a related algorithm, e.g., policy iteration) on the object-oriented representation.

The generation of the object-oriented state-transition diagram can be accomplished in a rule-based fashion or according to a “flat” representation (an appropriate explicit specification of the MDP structure). By rule-based, we mean any function that arranges the generation of the object-oriented state-transition diagram according to a set of rules as they are described for the simple instruction problem in Figure 3.3. Once the generation function realizing the corresponding set of rules is implemented in a generic way, it can also be used to generate object-oriented state-transition diagrams for instruction problems with more than two instructions[14].

[14]In principle, object-oriented state-transition diagrams for instruction problems with an arbitrary natural number of

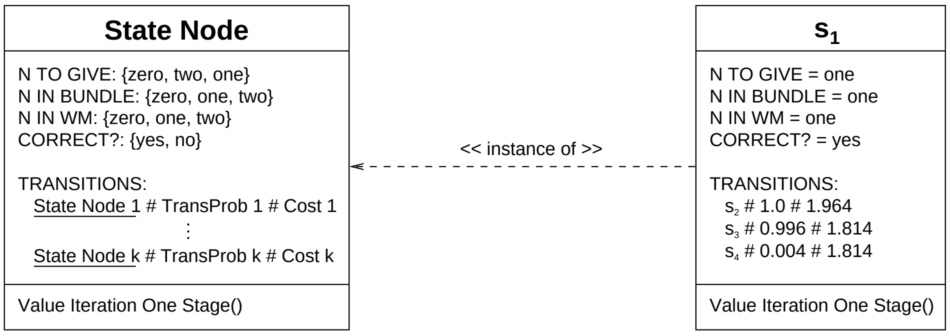

State Node
N TO GIVE: {zero, two, one}
N IN BUNDLE: {zero, one, two}
N IN WM: {zero, one, two}
CORRECT?: {yes, no}
TRANSITIONS:
State Node 1 # TransProb 1 # Cost 1
State Node k # TransProb k # Cost k
Value Iteration One Stage()
<< instance of >>
s1
N TO GIVE = one
N IN BUNDLE = one
N IN WM = one
CORRECT? = yes
TRANSITIONS:
s2 # 1.0 # 1.964
s3 # 0.996 # 1.814
s4 # 0.004 # 1.814
Value Iteration One Stage()

***Figure 3.11**: State node used in the model for the instruction problem: class and object.*
(The state node class comprises one attribute for each state feature N TO GIVE*,* N IN BUNDLE*,* N IN WM*, and* CORRECT?*. The attribute* TRANSITIONS *basically contains a list of triples describing the transitions from the state node instance at hand. There are transitions from s_1 to s_2, s_3, and s_4 with transition probabilities and costs as specifi ed in Figure3.4. The main operation of the state node class is the function implementing the computation of one iteration of the value iteration algorithm.*

In other domains, the structure of an MDP is not (exclusively) defined by a set of rules, but rather depends on characteristics of the domain environment. Spatial information, for example, the position of a user, can be relevant in a human-computer interaction process. The airport scenario in Section 4.2 is an example for such a domain. The specification of the relevant spatial information for a process, for example, the definition of important locations for giving navigation recommendations within a building, will usually be handcrafted. However, the required effort is independent of the selected representation formalism, that is, neither SPUDD nor the object-oriented approach described below can take advantage of the characteristics of such scenarios.

The application of the value iteration algorithm on the object-oriented representation can be split up in three phases: (1) initializing the value of the states, in particular of the goal states, (2) simultaneously computing the value function for the next stage, and (3) simultaneously updating the values of all states. The first phase basically means that the value of all non-goal states and undesired goal states is set to 0.0, and the value of the desired goal state(s) is set to positive reward(s). The values of the goal states do not need to be updated during phase two and three of the value iteration algorithm. In phase two, each state node object computes the expected utilities of all actions applicable in the corresponding state. Each state node gets the values of the successor[15] state nodes (the successor state nodes are known locally, as there exist pointers to them in the TRANSITIONS attribute) via message passing—a standard communication pattern between objects, which is based on remote method invocation (i.e., object A calls a method of object B and is passed the result). That is, the values of the successor states, as well as the transition probabilities and costs, are locally available in each state node object, such that each state node object can compute a part of the value function locally (cf. Equation 3.3). It is important to notice that the value of each node must not be updated before each state node has computed its value at the next stage. Otherwise, a state node might use values of adjacent states of different stages. Therefore, we split the computation of one iteration step in two phases—first computing the values

instructions could be considered. But computational resources impose an upper bound in practice.

[15]We speak of successor states, as the simple instruction problem has no cycles; in the general case, that is, if an MDP contains cycles, one has to speak of adjacent nodes or nodes in the direct neighborhood.

of all states locally (phase two), and then update the values of all states (phase three).

For the (simple) instruction problem, phase two and three of the object-oriented value iteration algorithm are repeated for a fixed number of times that equals the finite horizon n. In the case of an infinite-horizon problem, phase two and three are repeated until the difference between the values of two consecutive stages is sufficiently small in all state nodes.

3.1.5.3 Object-Oriented Approach Versus SPUDD

The last sections reveal advantages and disadvantages of both SPUDD and the object-oriented approach. Both approaches try to capture as much as possible of the structure inherent to the process to be planned. The object-oriented approach achieves a one-to-one mapping of the problem structure, as the state-based nature of decision-theoretic planning with MDPs perfectly matches the object-oriented programming paradigm. In fact, the decision for the extensional representation on which SPUDD is based brings about the need to put much emphasis on keeping at least part of the structure inherent to the problem. In this sense, the object-oriented approach appears to be the more "natural" approach if one puts emphasis on keeping as much structure as possible.

A clear advantage of SPUDD over the object-oriented approach is its generality. While SPUDD allows in principle to specify arbitrary MDPs (constrained only by the number of state features), the object-oriented approach is problem-tailored and requires low-level programming. However, the manual specification of problems in the format understandable by SPUDD is hardly feasible except for very small problems[16]. This means, that building a generator for the SPUDD representation of a problem requires some low-level programming, too. And in fact, the low-level programming required with both approaches is similar, as in both cases, it is supposed to achieve the automatic generation of an MDP representation, for example, according to a set o rules describing the structure of the MDP. In the case of SPUDD, the result is a specification of the MDP that SPUDD can understand, in the case of the object-oriented approach, the result is an object-oriented MDP.

Experiments with larger instances of the instruction problem on identical hardware indicate that the required computing time with the object-oriented approach is at least not worse than with SPUDD. Policy visualization and exploration come for free with SPUDD, but they can also easily be incorporated into the objects-oriented approach.

Considering all aspects, the object-oriented approach appears to be a competitive alternative to SPUDD. All problems that we describe in the following chapters were solved with the object-oriented approach. Our programming language of choice was the *Mozart Programming System*[17], which beyond other programming paradigms supports object-oriented programming. Table 3.5 summarizes the characteristics of SPUDD and the object-oriented approach.

3.1.6 Partially Observable Systems

So far, we have considered processes in which only the outcome of actions can be uncertain. There is a second type of uncertainty that can be relevant in decision processes. A system may not know the world state exactly when making its decisions on how to act, that is, it may have to rely on imprecise and noisy information and therefore have to base its choice of action on a

[16]Note that this is not in contradiction with the size of the problems (as stated in Section 3.1.5.1) that SPUDD can handle once they are specifi ed.

[17]Mozart is documented at and can be downloaded from `http://www.mozart-oz.org/`.

	SPUDD	OO-approach
representation	extensional, based on algebraic decision diagrams (ADDs)	intensional, based on objects-oriented programming paradigm
applicability	general Markov decision problems (constrained by the problem size)	tailored to specifi c problem (constraint by the problem size)
MDP specifi cation	ADD representations of action effects and rewards; illegible and complex for all but very small problems	rules defi ning the generation of the object-oriented state-transition diagram
capability of capturing problem structure	cannot always guarantee to capture the complete structure inherent to the problem	naturally captures the complete structure inherent to the problem
policy visualization/ exploration	provides general visualization and exploration tools	provide problem-tailored visualization and exploration facilities where necessary

***Table 3.5**: Comparison: Main characteristics of SPUDD and the object-oriented approach.*

probabilistic estimate of the world state (see, e.g., Drake, 1962; Astr¨om, 1965; Sondik, 1971; Smallwood & Sondik, 1973). This characteristic is called the *partial observability* of a process. To give an example of partial observability of a process, we can extend the simple instruction problem in the following way: Suppose, the system cannot directly determine the correctness of the execution of instructions, but each time the subject has executed one ore more instructions, the subject—instead of only confirming the execution by pressing the okay button—informs the system about if he thinks that he has executed the instruction(s) correctly or not. In this case, the system would have only uncertain information about the world state as the subject might think that he has executed a sequence of instructions correctly but (with a particular probability) in fact has not.

The partially observable extension of a Markov decision process is called a *partially observable Markov decision process (POMDP)*. Formally, the MDP framework is extended by a finite set of observations and an observation function.

Definition 3.3 *A* partially observable Markov decision process *is a tuple* $< \mathcal{S}, \mathcal{A}, \Omega, T, O, C, R >$, *where*

- $\mathcal{S}, \mathcal{A}, T, C, R$ *describe a Markov decision process;*
- Ω *is a finite set of* observations *about the world state; and*
- $O : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ *is the* observation function, *which gives, for each action action and resulting state, a probability distribution over possible observations; we write* $O(s', a, o)$ *for the probability of making observation o given that the system took action* a *and the world state changed to* s'.

Kaelbling et al. (1998) give an overview of the POMDP framework. The main difference for a system acting on a partially observable environment compared to a fully observable environment is that instead of keeping track of the exact world state, the system keeps an internal *belief state*. The belief state is a probability distribution over states of the world and summarizes the system's complete experience in the environment. That is, given the current belief state, no additional data about past actions or observations would supply any further information about the current world state. A system acting on a partially observable environment needs basically two components: a *state estimator* (which updates the belief state-based on the last action, the current observation, and the previous belief state) and a policy that provides the optimal next action based on the system's belief state (rather than on the exact world state as in a fully observable environment).

For both exact and approximate algorithms to solve POMDPs, different new approaches have been studied during the last decade. Zhang and Liu (1996) and Cassandra, Littman, and Zhang (1997) describe the *incremental pruning* algorithm. Like preceding approaches, incremental pruning uses a form of dynamic programming in which a piecewise-linear and convex representation of one value function is transformed into another. According to the results presented by Cassandra et al. (1997), incremental pruning outperforms other algorithms like, for example, the witness algorithm introduced by Littman (1994). However, Kaelbling et al. (1998) describe the *witness*[18] *algorithm* as one of the most efficient algorithms to compute exact solutions for POMDPs. Experimental results (Littman, Cassandra, & Kaelbling, 1995) suggest that the witness algorithm becomes impractical for problems of modest size (e.g., $\mid \mathcal{S} \mid > 15$ and $\mid \Omega \mid > 15$), and incremental pruning does not outperform the witness algorithm in orders of magnitude. Moreover, all problems considered for comparison are at best of a moderate size (e.g., 12 states, 6 actions, 5 observations, 4 stages).

Approximate algorithms for POMDPs have in particular been analyzed in the context of robot navigation (e.g., Roy & Thrun, 1999; Thrun, 2000) and recently also for spoken dialog management (e.g., Roy et al., 2000; Montemerlo, Pineau, Roy, Thrun, & Verma, 2002—cf. also Section 2.2.7.2). The approximation is based on the idea that the *most likely world state* in addition with the *entropy* of the belief state provides a sufficient statistic for the entire belief state. That is, a policy is computed for every $\{state, entropy\}$-pair instead of the continuous belief state. Pineau, Roy, and Thrun (2001) and Pineau, Gordon, and Thrun (2003) describe a hierarchical approach to POMDPs and among other domains test it for spoken dialog management.

Several other approaches to compute approximate solutions for POMDPs have been proposed. For example, Hansen (1998) introduces an approach that represents a policy explicitly as a finite-state controller and iteratively improves it by searching in policy space. A new heuristic search algorithm focuses the computational effort on regions of the problem space that are reachable or likely to be reached from a start state. Ng and Jordan (2000) propose a method that transforms a POMDP into an "equivalent" POMDP in which all state transitions are deterministic and then search for a policy with high estimated value. Hansen and Feng (2000) describe an approach using a factored state representation, and thereby extend the framework introduced by Boutilier and Poole (1996). Brafman (1997), Hauskrecht (2000), Zhang and Zhang (2001), Zhou and Hansen (2001) and others describe *grid-based approximation* methods for POMDPs. The basic idea is to place a finite grid over the *belief simplex*[19], compute values for points in the grid, and interpolate to evaluate all other points in the simplex.

This section indicates the particular complexity problems involved in solving POMDPs. We will summarize some of the important general aspects about complexity involved in solving both fully observable and partially observable Markov decision processes in the next section.

[18]The name "witness" refers to the technique that is used to determine when an exact representation of the value function is found.

[19]A simplex $S \subset \mathbb{R}^n$ is the convex hull of $n + 1$ points P_i that do not all belong to the same hyper plane; for example, a two-dimensional simplex is a triangle, a three-dimensional simplex is a tetrahedron.

3.1.7 About the Complexity of Solving MDPs and POMDPs

The question if the techniques considered in this chapter scale to solve planning problems of reasonable size has to be analyzed from two perspectives: (1) by considering the complexity of the state space and (2) by considering the complexity of policy construction. We will comment on the results for both questions in the light of the problems that we will consider in the context of user-adaptive systems.

3.1.7.1 State Space Complexity

We have already mentioned *Bellman's curse of dimensionality* in earlier sections: The state space of an FOMDP grows exponentially with the number of state features. Using a general approach for arbitrary Markov decision problems (e.g., SPUDD), one has to arrange for an algorithm working on a factored representation of the state space to focus computational effort on reachable parts of the state space (e.g., as supported by the use of ADDs in SPUDD). Hand-crafted or automatically generated, problem-tailored models ensure—assuming a proper construction—that only relevant states are considered by any algorithm working on such explicit representation. However, in the worst case, all combinations of the state features' values represent a relevant (reachable) state, that is, neither the factored nor the explicit representation of the state space overcomes Bellman's curse of dimensionality. The simple instruction problem as well as the examples of the forthcoming chapters illustrate that decision processes in the context of user-adaptive systems are often sufficiently well structured to allow the consideration of problems (and problem sizes) relevant in practice.

In the context of POMDPs, the state space problem is aggravated: Instead of considering a discrete (world) state space, one faces a continuous (i.e., infinite) belief state space. However, all approaches to POMDPs exploit the insight that the finite-horizon value function over the belief space is piecewise linear and convex for every horizon length. That is, for each iteration of the value iteration algorithm, it is required to find only a finite number of linear segments that make up the value function. Problems of the size of a partially observable instance of the simple instruction problem—one might say toy problems—can in principle be tackled using any of the exact approaches described in Section 3.1.6.

Work on approximate methods, for example, as proposed by Roy et al. (2000), promises to allow addressing at least some interesting partially observable problems that might arise from our domains of interest in the future. By compressing the belief state to the most likely state and the entropy of the belief state, they solve problems that involve 20 actions, 16 observations, and 13 world states. However, the authors comment only on the time needed to solve a considerably smaller test problem.

For solving partially observable Markov decision processes, Poupart and Boutillier (2004) identify two major sources of intractability: (1) the curse of dimensionality and (2) the policy space complexity. With their VDCBPI[20] algorithm, they describe the first approach to tackle both these sources of intractability and can demonstrate the scalability of the VDCBPI algorithm on synthetic network management problems with up to 33 million states. Thus, the VDCBPI algorithm might allow to use POMDP models in the context of user-adaptive system in the future.

[20]VDCBPI stands for the **V**alue **D**irected **C**ompression technique with **B**ounded **P**olicy **I**teration.

3.1.7.2 Complexity of Policy Construction

FOMDPs with time-separable, additive value functions can be solved in time polynomial in the size of the state space, the number of actions, and the size of the inputs[21] (Boutilier et al., 1999). Finite-horizon and discounted infinite-horizon problems require a polynomial amount of computation per iteration—$O(\mid S\mid^2\mid A\mid)$ and $O(\mid S\mid^2\mid A\mid + \mid S\mid^3)$, respectively—and converge in a polynomial number of iterations. This explains why the size of the state space that has to be considered is crucial for the time required to solve an FOMDP. The space required to store the policy for an n-stage finite-horizon problem is $O(n\mid S\mid)$, the space required for the stationary policy of an infinite-horizon problem is $O(\mid S\mid)$. Further complexity results for FOMDPs are summarized by Littman, Dean, and Kaelbling (1995).

The problem of finding an optimal policy for a POMDP with the objective of maximizing expected total reward or expected total discounted reward over a finite horizon t is exponentially hard both in $\mid S\mid$ and t (Papadimitriou & Tsitsiklis, 1987). The problem of finding a policy that maximizes or approximately maximizes the expected discounted total reward over an infinite horizon is shown to be undecidable (Madani, Hanks, & Condon, 1999). Further nonapproximability results for POMDPs are summarized by Lusena, Goldsmith, and Mundhenk (2001).

3.2 Synopsis

In this chapter, we have surveyed the elements of Markov decision processes[22] and the standard algorithms to solve them. We have illustrated the basic concepts with a simple example from an experimental setting, and we have emphasized on different possibilities to represent an MDP. The analysis of two approaches, an off-the-shelf approach based on a factored representation and a domain-tailored approach based on object-oriented programming, has revealed that the latter is more appropriate for the problems that we consider in the context of user-adaptive systems. Moreover, an empirical comparison of the two approaches (based on the simple instruction problem that we have used throughout this chapter) has shown that our approach is about ten times faster than the approach based on the factored state space representation.

We have briefly looked at partially observable Markov decision processes, which would have allowed to consider problems in user-adaptive systems that involve uncertain feedback about the state of the interaction between the system and the user. Unfortunately, the problem of how to solve large POMDPs efficiently is still an open question that keeps the AI planning community busy. As the remaining chapters will show, there exist manifold problems of relevance in the field of user-adaptive systems for which it is appropriate to assume full observability of the state of interaction. In the next chapter, we apply the object-oriented approach to moderately sized Markov decision problems of practical relevance for user-adaptive systems.

[21]More precisely, the maximum number of bits required to represent any of the transition probabilities or costs.

[22]Closely related to Markov decision processes are hidden Markov models (see, e.g., Rabiner, 1989). A *hidden Markov model* (HMM) is a statistical model where the system being modeled is assumed to be a Markov process with unknown parameters. Based on this assumption, the challenge is to determine the hidden parameters from the observable parameters. The extracted model parameters can then be used to perform further analysis, for example, for pattern recognition applications. Prominent application areas of HMMs are, for example, speech recognition, optical character recognition, natural language understanding, bioinformatics, and genomics.

4 FOMDPs in Scenarios of Low Complexity

In the previous chapter, we have introduced the basic concepts of decision-theoretic planning and illustrated its application with an example based on a simple form of interaction between a system and its user. In this chapter, we will describe two examples for the application of decision-theoretic planning in realistic contexts. We will consider one problem in the context of a stationary application, in which a system provides adaptive assistance for operating a technical device, and one problem in the context of a mobile application, in which the system provides adaptive navigation recommendations within a complex building. To make the examples more concrete, we place both applications in an airport scenario: We will describe an airport assistance system that (1) helps the user with instructions on how to operate a credit card phone and (2) provides the user with adaptive navigation recommendations to get to the gate and potentially buy something in one of the stores along the way.

4.1 Assistance for Operating a Technical Device

Many technical devices provide some kind of build-in assistance, for example, for situations in which the user needs some information before he can proceed with a task (e.g., the DT Tutor system; see Section 2.1.3.5) or to make the user aware of ways to do a task that he is already familiar with more efficiently (e.g., the Microsoft Office Assistant; see Section 2.1.3.2). The case that we consider in this section is different. We analyze a situation in which an assistance system $\mathcal{S}$ explains to the user $\mathcal{U}$ how to operate a credit card phone that is installed somewhere in an airport building. That is, the assistance functionality that we consider is not build-in to the system (i.e., the credit card phone) itself, but the system acts as an external assistant for a technical device. We assume that this assistance system runs on a small handheld computer that provides input and output facilities like pen, microphone, display, loudspeaker, and the like.

4.1.1 External Assistance Systems for Technical Devices

In this section, we come back to the WIP system, which we have already described in Section 2.2.2.2, and compare WIP with the READY assistance system for operating a credit card phone. As a further example of assistance systems for technical devices, we describe ADAPTS, an assistance system for maintenance technicians in the helicopter domain.

4.1.1.1 Wahlster et al. (1993): WIP (revisited)

In Section 2.2.2.2, we have reviewed the multimodal presentation system WIP (Wahlster et al., 1993). To illustrate its functionality, it was applied for providing assistance to operate an espresso machine, to assemble a lawn-mower, and to maintain a modem. In contrast to the READY assistance system for operating a credit card phone, WIP plans presentations incrementally, that is, all presentation decisions are postponed until runtime. With regard to the limited computational resources available on many handheld computers, complex decision making at runtime can be disadvantageous.

The decision-theoretic planning approach yields a policy as the solution to a planning problem. As a policy can be computed offline, for example, on the workstation of an information kiosk or on the personal computer at home, the handheld device merely needs to apply the policy like a look-up table. While interactivity is not a strength of the WIP system (a drawback that was eliminated in the PPP system based on WIP—see Section 2.2.2.3), the application of a policy allows for some interactivity as far as the user's actions are foreseeable and considered in the decision-theoretic model. In restricted domains, such as the ones that WIP was used for, a decision-theoretic model can cover the potential variations of the interaction between the system and its user to a great extend.

WIP coordinates output in different modes. In the READY assistance system for operating a credit card phone, we have not taken different forms of presentation into account. However, different forms of presentation for this application can be considered in the decision-theoretic model in analogy to the model described in Section 4.2.3. Different forms of presentation will be discussed in more detail in Section 4.1.3.1.

4.1.1.2 Brusilovsky & Cooper (2002): ADAPTS

ADAPTS[1] (Brusilovsky & Cooper, 2002), an electronic performance support system (EPSS) for maintenance technicians, integrates adaptive guidance from diagnostics systems with adaptive access to technical information. ADAPTS adjusts the diagnostic strategy to who the technician is and what the technician is doing, dynamically adapting the sequence of setups, tests, and repair/replace procedures based on the technician's responses. New activities are planned depending on the technician's responses to current recommended activities. The system assembles information content on the fly in response to the steps of that diagnostic process. The technician receives dynamically selected technical support information appropriate for the the setup, test, and remove/replace procedure being performed. The user model determines what task to do, what technical information to select to describe the task, and how to best display that information for a given technician.

Typical hypermedia systems identify a predefined course through technical information. The ADAPTS system, on the other hand, dynamically defines a unique course each time it presents technical information. In this case, the adaptive diagnostics components serves as the expert technician, driving the troubleshooting strategy based on a dynamic assessment of time, effort, payoff, resources, and a specific technician's knowledge and experience with a specific troubleshooting scenario.

Like in an earlier version of the READY system, ADAPTS bases its adaptation strategy exclusively on the user model and the assessment of the situation at the time of decision making. In

[1]Possible scenarios for using adaptive hypermedia for adaptive performance support were explored in the context of the ADAPTS (Adaptive Diagnostics and Personalized Technical Support) project.

the scenario at hand, we would expect that it is worthwhile to take potential ways of how the troubleshooting process might evolve in the future into account, and thereby guide the troubleshooting that the technician performs into the right direction (i.e., such that it can be completed effectively).

In the following sections, we will illustrate how decision-theoretic planning can be applied in the context of an external assistance systems for operating a credit card phone.

4.1.2 Example Domain

Although operating a credit card phone is not a very difficult task in general, people get frequently confused about the procedure under certain circumstances. The unfamiliarity with a particular type of credit card phone stands to reason, for example, if the user is a traveler who has just arrived in a foreign country. A similar problem arises, for example, in the context of operating a ticket machine (for bus or train tickets) or operating the petrol pump of a filling station (especially those that offer different sorts of petrol at the same petrol pump and allow to pay by credit card at the tap directly). Not all systems that should provide assistance in fact do provide (sufficient) assistance. The lack of language ability enforces the need for external assistance—yet, the problem exists also for people with adequate knowledge of a language. Therefore, providing external assistance for such devices is a reasonable attempt to help users and to offer a valuable service to them.

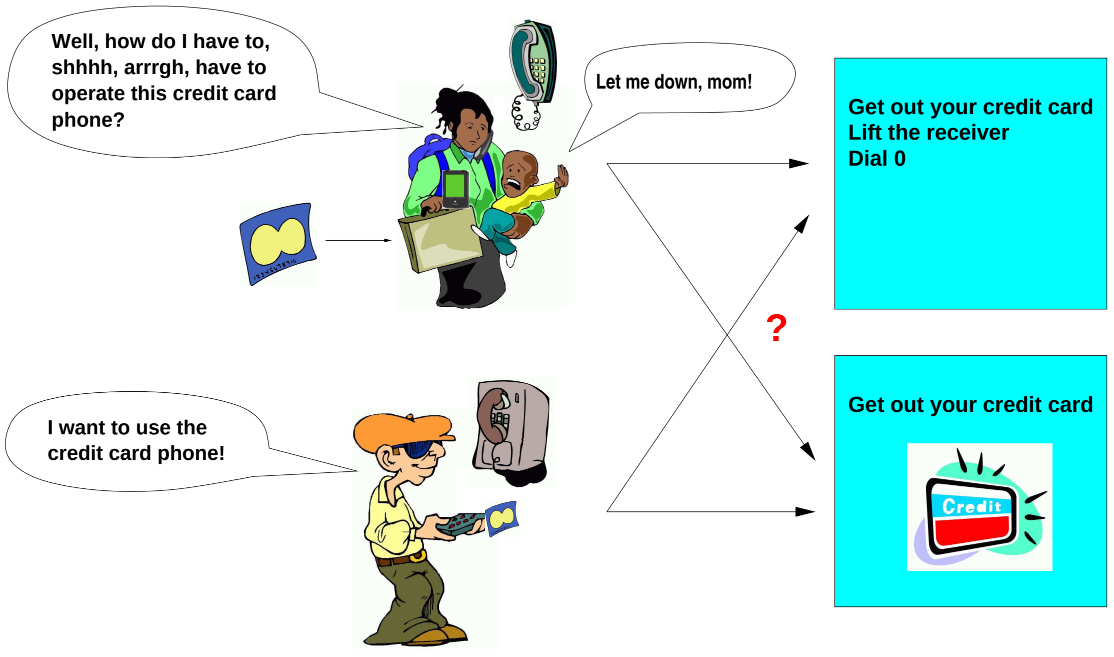

Well, how do I have to, shhhh, arrrgh, have to operate this credit card phone?
Let me down, mom!
Get out your credit card
Lift the receiver
Dial 0
?
I want to use the credit card phone!
Get out your credit card
Credit

***Figure 4.1**: Assistance for operating a credit card phone in two different situations.*
(A user-adaptive system should take the user's situation into account when making the decision about how to present information to the user.)

The ability to operate a technical device is not only dependent on static preconditions (such as the knowledge of a language) but also on the user's current situation. Figure 4.1 illustrates two situations that require different strategies by $\mathcal{S}$ to provide assistance for $\mathcal{U}$. The woman who carries her luggage and child is distracted by the kid's complaints and will hardly be able to concentrate and follow several instructions given in one turn. Short and simple instructions, possibly illustrated with meaningful pictures, might be more appropriate for her. The young man, who apparently has already got rid of his luggage, seems to be relaxed and can fully concentrate on operating the phone. For him, the bundling of several instructions might be more appropriate. He might even be annoyed if being assisted with unnecessarily verbose instructions (cf. Section 1.2; dialog between a customer and an ATM). We use the decision-theoretic planning approach in particular to achieve adaptation to situative resource restrictions (see also Bohnenberger, 2001).

4.1.3 Assumptions

Operating a credit card phone requires a sequence of actions starting with holding the credit card ready and lifting the handset, dialing some preliminary digits before entering credit card information, and eventually dialing the desired number. $\mathcal{S}$ could for example use the following eight instructions to assist $\mathcal{U}$ during the operating process:

1. "Hold your credit card ready."
2. "Lift the handset."
3. "Dial 0."
4. "After the tone, dial 9."
5. "After the tone, enter your credit card number."
6. "Enter two digits for the month of expiration."
7. "Enter two digits for the year of expiration."
8. "After the tone, dial the desired number."

Even with this small set of instructions, a number of different instruction sequences can be generated by bundling several instructions together. Neither giving all of these instructions step by step, nor giving all of them in one turn seems to be the appropriate strategy in all situations. In some situations, the optimal strategy might lay somewhere in between, for example, bundling the first three instructions, giving two instructions stepwise, bundling two instructions, and giving the last instruction stepwise again. The bundling strategy depends also on the complexity of individual instructions.

4.1.3.1 Variations of the Form of Output Presentation

The number of possible variations grows if we consider different forms of presentation and syntactical variations of the same content. The instruction "Hold your credit card ready." can, for example, be prompted on the display of the handheld computer, but it can also be given in spoken language by playing the corresponding sound file on the handheld computer, or it can be visualized, for example, by an appropriate illustration or animation on the display of the handheld computer. Also combinations of different forms of output presentation can be appropriate in some situations. Using the same form of output presentation, there still exist reasonable variations. For example, instead of the utterance "Hold your credit card ready." the more verbose version "You will need information from your credit card to use this phone. Please, hold the credit card ready." can be more appropriate under certain circumstances (e.g., to reassure that $\mathcal{U}$ knows that the phone he wants to use works with credit card).

All of the mentioned options have advantages and disadvantages. For example, speech output does not require $\mathcal{U}$ to hold the handheld computer in a position such that he can read the output on the display. On the other hand, speech output might be error prone in a noisy environment. Advantages and disadvantages of this kind can be identified for each form of output that we have listed. With the Markov decision process that we describe in Section 4.1.4, we will focus only on the bundling of instructions and not on variations of the form of output presentation or syntactical variations of the same content. Different forms of output presentations are used in the application described in Section 4.2.2: The system described there can use different forms of presentation, for example, one providing only the basic navigation information and one providing additional information about the close environment.

4.1.3.2 Feedback About the Interaction

As $\mathcal{S}$ is an external help system—$\mathcal{S}$ is not integrated in the device to operate and therefore has no means to find out about $\mathcal{U}$'s performance directly—$\mathcal{U}$ has to give feedback explicitly whenever he is finished with an instruction. He could, for example, press a push-button (as they can often be found below the display of a handheld computer), tab with a pen[2] on an okay button on the display of the handheld computer or say something like "Okay, I've done this..." if the system can deal with speech input. However, giving feedback takes time and is annoying for $\mathcal{U}$ if required unnecessarily often. Instead, $\mathcal{S}$ should bundle instructions in the most convenient way for $\mathcal{U}$'s current situation and thereby minimize the number of times $\mathcal{U}$ is required to give feedback.

We assume that $\mathcal{U}$'s feedback, no matter what form of input $\mathcal{U}$ chooses, is fully observable for $\mathcal{S}$. If $\mathcal{U}$, for example, says "Okay, I've done this...", this means that $\mathcal{U}$ was able to execute $\mathcal{S}$'s last instruction (or bundle of instructions) completely and correctly. Note that this model does not consider the case in which $\mathcal{U}$ has completed one or several instructions, thinks that he has completed all of them correctly, but in fact has made one or several mistakes when following $\mathcal{S}$'s instructions. Considering this case would involve a partially observable model, in which $\mathcal{U}$ might provide uncertain feedback about the state of the interaction in terms of observations that he might report to the system. Uncertainty about the state of the world often results from the user not being able to determine exactly what problem he faces. For example, after having inserted the credit card, $\mathcal{U}$ might say something like "I can hear a sequence of short tones, now...", by which $\mathcal{S}$ might infer that with a certain probability $\mathcal{U}$ has inserted the credit card incorrectly, that $\mathcal{U}$ has executed two instructions in the wrong order, that the credit card phone is out of order, or something similar.

However, we assume that $\mathcal{U}$ provides reliable and sufficient feedback for $\mathcal{S}$ to assume the interaction to be fully observable. If $\mathcal{U}$ is, for example, uncertain about the order in which he is supposed to execute two instructions, we assume that he would ask for repetition of the last instruction or instruction bundle. How a repeat action can be realized and what effect it has on the bundling strategy of the system will be described in Section 4.1.4.1.

[2]Most of today's handheld computers are mainly operated with a pen. However, the form of input is secondary in this context.

4.1.4 Model

The model that we have developed for the experimental setting in Section 3.1.1, and by which we have explained the decision-theoretic planning approach, can directly be transfered to the example at hand—giving adaptive assistance for a technical device. Figure 4.2, which describes the structure of the Markov decision process to plan the dialog between $\mathcal{S}$ and $\mathcal{U}$, is more or less a blueprint of the structure of the MDP to plan the interaction in the experimental setting (cf. the explanation of Figure 3.3 for more details about the structure of the decision-theoretic model at hand.).

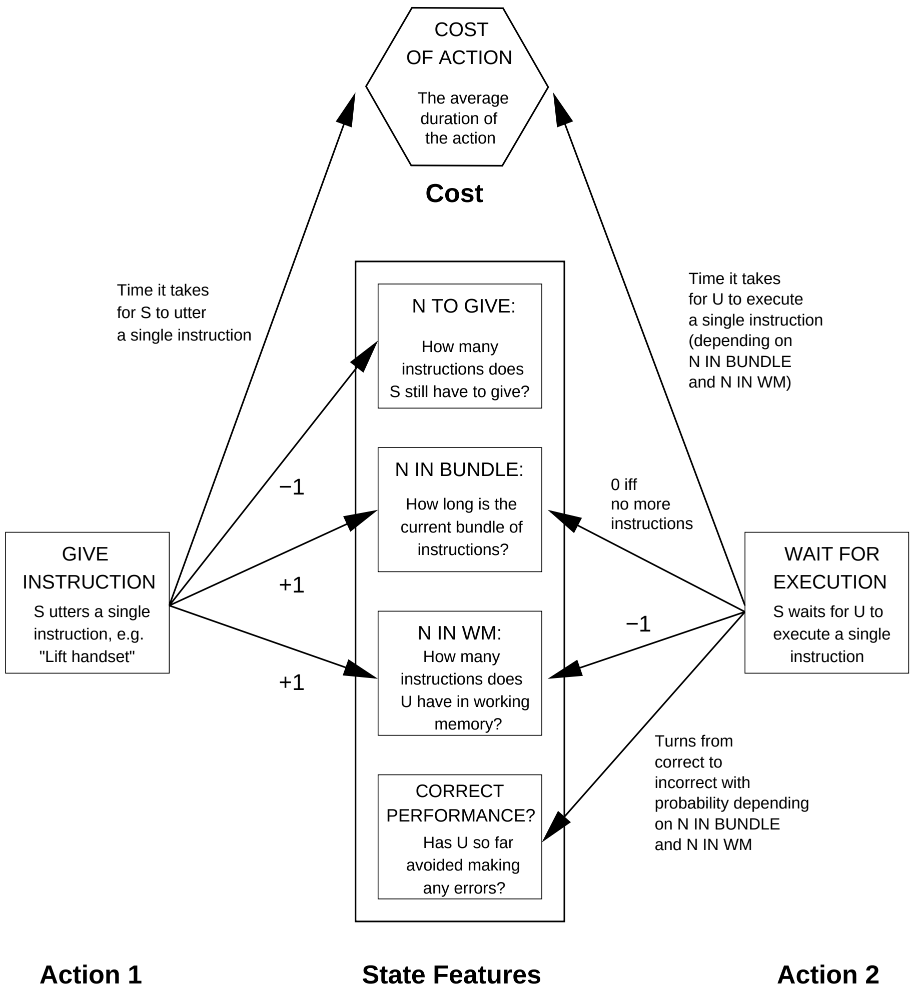

COST
OF ACTION
The average duration of the action
Cost
Time it takes for S to utter a single instruction
Time it takes for U to execute a single instruction (depending on N IN BUNDLE and N IN WM)
N TO GIVE:
How many instructions does S still have to give?
N IN BUNDLE:
How long is the current bundle of instructions?
N IN WM:
How many instructions does U have in working memory?
CORRECT PERFORMANCE?
Has U so far avoided making any errors?
−1
+1
+1
0 iff no more instructions
−1
Turns from correct to incorrect with probability depending on N IN BUNDLE and N IN WM
GIVE INSTRUCTION
S utters a single instruction, e.g. "Lift handset"
WAIT FOR EXECUTION
S waits for U to execute a single instruction
Action 1
State Features
Action 2

Figure 4.2: *Structure of the MDP to plan the instructions for operating a credit card phone. (The structure for the given problem is identical to the structure in the experimental setting (cf. Figure 3.3). The difference is merely of a quantitative nature and cannot be seen in this fi gure. The same MDP structure could be used for similar problems in other contexts, too.)*

The construction of the MDP starts with a state in which the variable N TO GIVE equals the number of instructions that are to be given. If we want to plan the complete sequence of instructions, N TO GIVE equals 8. The variables N IN BUNDLE and N IN WM equal 0 in the initial state, the variable CORRECT PERFORMANCE? equals *true*. The only possible action in this state is GIVE INSTRUCTION (“Get out your credit card”). The corresponding transition is deterministic and leads to the state where N TO GIVE equals 7, N IN BUNDLE and N IN WM equal 1, and CORRECT PERFORMANCE? is still *true*. From this state, $\mathcal{S}$ can either give the next instruction or wait for $\mathcal{U}$ to execute the instruction already given by $\mathcal{S}$. The transition of the action GIVE INSTRUCTION from this state has the same effects as in the initial state: N TO GIVE is decreased by 1, N IN BUNDLE and N IN WM are increased by 1. The action WAIT FOR EXECUTION brings about two transitions. One leads to a state where CORRECT PERFORMANCE? is still *true*, the other leads to a state where CORRECT PERFORMANCE? is *false*—the user has made a mistake when executing the instruction, for example, he has taken out the wrong credit card. In both successor states, the variable N TO GIVE has not changed its value compared to its predecessor state, but N IN WM has been decreased by 1 and N IN BUNDLE has been set to 0. The two transitions are annotated with the probabilities that $\mathcal{U}$ executes the instruction (given a bundling size of 1 and 1 instruction in his working memory) correctly or with a mistake, respectively. The procedure for creating new states and transitions for the two possible actions is repeated until all new transitions end in goal states (where N TO GIVE and N IN WM both equal 0).

While the structure of the resulting MDP is the same as the structure of the MDP for the simple instruction problem (see Section 3.1.2), the costs, that is, each of the durations of $\mathcal{S}$ giving a particular instruction and $\mathcal{U}$ executing a particular instruction, as well as each of the probabilities of $\mathcal{U}$ making a mistake when trying to execute a particular instruction, have to be adjusted. As in the experimental setting of Section 3.1.1, the time needed by $\mathcal{S}$ to give an instruction can be measured directly, the time needed by $\mathcal{U}$ to execute a particular instruction can be determined by averaging the times needed in a number of test cases, and the probability that $\mathcal{U}$ executes a particular instruction correctly can be derived from the relative frequency of correct executions of a particular instruction in a number of test cases.

4.1.4.1 An Additional Repeat Action

Bohnenberger (2000) has studied the effects of introducing an additional action REPEAT INSTRUCTION in the context of the experiment described in Section 3.1.1. The idea is that the user can ask the system to repeat the last instruction(s) if he was not able to execute them correctly. Thereby, we assume that the system can only repeat a fixed number of instructions. This assumption requires an additional state feature in the decision-theoretic model to keep track of the remaining number of repetitions that the system is allowed to do.

In the experimental setting, we have studied the policies for giving 4 instructions with variations in the following parameters: (1) the reward for executing all 4 instructions correctly, (2) the information if the user is distracted or not, and (3) the maximal number of allowed repetitions. The rewards reflect the importance of executing all instructions correctly. A reward can be interpreted as having an additional amount of time available when reaching the goal. For example, a reward of 10000 means that executing all instructions correctly is worth for the user as much as having some additional 10000msec (or 10sec) available. Against this background, the results shown in Figure 4.3 must be interpreted as follows: For example, if the user is not distracted and the system is not allowed to repeat any instruction, then the dominant strategy[3] is to bundle two times two instructions together. If the system is allowed to repeat one instruction, the dominant strategy is to give all instructions stepwise—the policy recommends other strategies only for very small rewards. If the system is allowed to repeat two instructions, the dominant strategy is to

[3]We use the term *dominant strategy* in an informal way; it means that this strategy is chosen for nearly all rewards.

bundle two times two instructions together if executing all instructions correctly is worth less than about 12.4sec of additional time, to give the first two instructions stepwise and bundle the last two instructions if executing all instructions correctly is worth between about 12.4sec and 24.2sec of additional time, and to give all instructions stepwise if executing all instructions correctly is worth more than about 24.2sec of additional time. The other table entries can be interpreted analogously.

		Maximal Number of Allowed Repetitions				
		0	1	2	3	4
No Distraction	one bundle of 4 instructions	0 − 5	0 − 5	0 − 4	−−−	−−−
	two bundles of 2 instructions	6 − infinite	6 − 176	5 − 12384	0 − 45042	0 − 99999
	one bundle of 2 instructions and 2 instructions stepwise	−−−	177 − 181	−−−	−−−	−−−
	2 instructions stepwise and one bundle of 2 instructions	−−−	−−−	12385 − 24153	45043 − 46099	−−−
	4 instructions stepwise	−−−	182 − 99999	24154 − 99999	47000 − 99999	−−−
Distraction	one bundle of 4 instructions	0 − 3	0 − 3	0 − 2	−−−	−−−
	two bundles of 2 instructions	4 − 16	4 − 24	3 − 323	0 − 736	0 − 14500
	one bundle of 2 instructions and 2 instructions stepwise	17 − 19	25 − 27	−−−	−−−	−−−
	2 instructions stepwise and one bundle of 2 instructions	−−−	−−−	324 − 378	737 − 741	14501 − 17133
	4 instructions stepwise	20 − 99999	28 − 99999	379 − 99999	742 − 99999	17134 − 99999

***Figure 4.3**: Bundling policies under varied parameters.*
(The table specifi es the reward intervals for which the system uses a particular instruction policy, given the information if the user is distracted or not and the number of instructions the system is allowed to repeat.)

Figure 4.3 gives a notion of how the assumption that the user can ask the system for repetition of instructions changes the overall instruction strategy of the system. In the READY prototype demonstrating the assistance for operating a credit card phone, the user is also allowed to ask for repetition of instructions. We have experimented with different degrees of look ahead and different numbers of allowed repetitions. The example of assisting the user when operating a credit card phone includes eight instructions altogether. Effective configurations for this example are to plan either the first three, then two, and the last three instructions, or to plan two times four instructions, each time allowing two or three instructions to be repeated if necessary. Dealing, for example, with two times four instructions in two planning processes instead of considering all eight instructions in one planning process facilitates the incorporation of updated information from the user modeling component during the interaction of the system with the user. In the following example of the system behavior, the user does not ask for repetition but manages to execute all instructions correctly in the first attempt. Appendix B lists a part of a sample trace of the computations for planning a sequence of three instructions with one repetition allowed.

4.1.5 System Behavior

In this section we describe the behavior of the system assisting a passenger who is in a hurry when he requests help from the airport assistance system. His articulation rate is high, and on his first attempt he does not click accurately on the phone icon displayed on his PDA's display, clicking slightly next to it instead. For the example, we assume the kind of environment that is described in Section 4.1.6, including a user modeling component based on a dynamic Bayesian network. Instantiation of the nodes that correspond to the above described symptoms in the dynamic Bayesian network causes $\mathcal{S}$ to infer a relatively high level of time pressure. The updated user model is now used to parameterize the decision-theoretic planning process. A dialog policy is computed, which determines the system's next four instructions. $\mathcal{S}$ starts with a single instruction:

- $\mathcal{S}$: "Get out your credit card."
- $\mathcal{S}$ waits for feedback (e.g., $\mathcal{S}$ might say something like "OK, I've got my credit card ready now . . . ").

$\mathcal{S}$ then bundles the next two instructions:

- $\mathcal{S}$: "Lift the receiver."
- $\mathcal{S}$: "Dial 0."
- $\mathcal{S}$ waits for feedback.

In principle, $\mathcal{S}$ could have given the first three instructions in one bundle. Many factors contributed to the decision to give the first one separately; one of them is the relatively long duration of the action of getting out the credit card, which decreases the likelihood that any subsequent instructions within the same bundle would be remembered correctly. Although $\mathcal{U}$'s articulation rate has meanwhile decreased to a moderate level, $\mathcal{S}$'s estimate of $\mathcal{U}$'s time pressure is still relatively high, since the dynamic Bayesian networks incorporate the assumption that the degree of time pressure is unlikely to change suddenly at any given moment. There is still no evidence for an unusual level of cognitive load.

$\mathcal{S}$ bundles the next two instructions again:

- $\mathcal{S}$: "After the tone, dial 9."
- $\mathcal{S}$: "After the tone, enter your credit card number."
- $\mathcal{S}$ waits for feedback.

These instructions would not be bundled if the actions were to be performed in the reverse order: After the complex action of entering the credit card number, the probability of remembering any other instruction would be relatively low.

$\mathcal{S}$ gives the last three instructions stepwise:

- $\mathcal{S}$: "Enter two digits for the month of expiration."
- $\mathcal{S}$ waits for feedback.
- $\mathcal{S}$: "Enter two digits for the year of expiration."
- $\mathcal{S}$ waits for feedback.
- $\mathcal{S}$: "After the tone, dial the desired number."
- $\mathcal{S}$ waits for feedback.

This choice of stepwise presentation is influenced in part by the working memory demands of the actions involved. For example, entering information about the expiration date is classified as being more demanding than lifting the receiver. It is possible to classify the inherent difficulty of executing a particular instruction as *simple*, *moderate*, or *difficult* in the READY prototype (see Section 4.1.6), which specifies the influence that the situative resource restrictions have on the probabilities used in the decision-theoretic planning process.

4.1.6 Assistance for Operating a Technical Device in the READY Prototype

With the READY prototype, we describe the type of environment in which decision-theoretic planning can be useful to adapt the system behavior to the user's current situation.

In Section 3.1.4 we have seen how a decision based on decision-theoretic planning depends on the reward for a desired goal state (cf. in particular Figures 3.5 and 3.6). A high reward for the desired goal state results in a policy that recommends giving the two instructions of the simple instruction problem stepwise, while a low reward results in a policy that recommends bundling the two instructions. In fact, the qualitative measures for the rewards, costs, and probabilities provide the interface to a system's user modeling component.

In the following subsections, we describe the individual components of the READY prototype and how they interact with each other. In particular, we will explain the relation between the user modeling component and the decision-theoretic planner.

4.1.6.1 The Architecture of the READY Prototype

We take a brief look at the architecture of the READY prototype (see also Bohnenberger, Brandherm, Großmann-Hutter, Heckmann, & Wittig, 2002) and in particular to its user modeling component before we explain the influence of the user model on the decision-theoretic planning process. Figure 4.4 summarizes the components of the READY prototype and indicates which components interact with each other.

We first survey all components of the complete READY architecture before we describe the particular relation between the Decision-Theoretic Planning component and the User Modeling component in Section 4.1.6.3. The two pass information via the so called Interaction Manager. The role of the Interaction Manager is to establish the connections to the Decision-Theoretic Planning component, the User Modeling component, and the User Interface, and to execute a script which determines the individual coarse steps of the interaction between the system and the user. Each of the components listed in Figure 4.4 runs in its own process. They interchange data via the Interaction Manager on the basis of socket communication—the interface specification is given in Appendix C.

The two architecture segments Experiments and BN Learning/BN Adaptation (upper half of Figure 4.4) do not belong to the runtime components of the READY prototype. The empirical data gained in Experiments are exploited in both the Decision-Theoretic Planning component and for the learning and adaptation of Bayesian networks (BN Learning/BN Adaptation; cf. Wittig, 2003). The Bayesian networks form the core of the User Modeling component.

User Modeling Bayesian networks (see Section 2.1.3.1) have gained increased popularity for making inferences in user modeling components of complex systems (see, e.g., Horvitz et al., 1998). In particular, two properties of Bayesian networks make them especially appropriate for

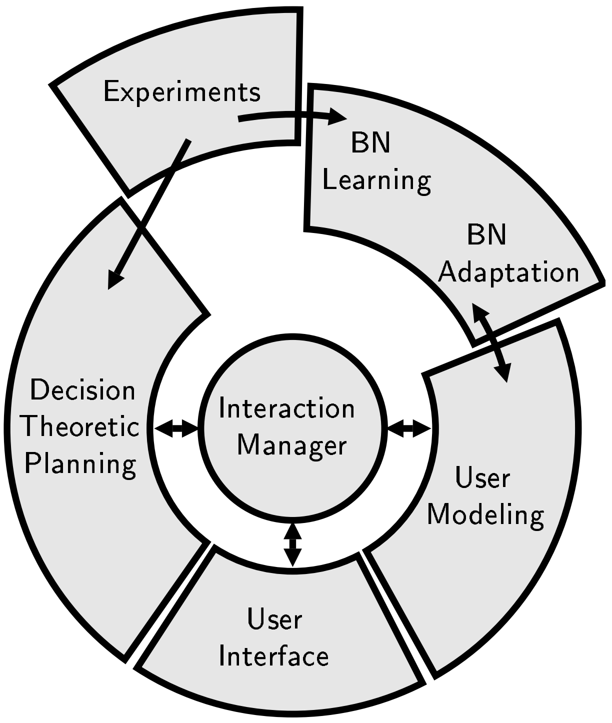


***Figure 4.4**: Components of the READY prototype.*
(The decision-theoretic planning component, the user modeling component, and the user interface communicate with each other via a so called interaction manager. Parts of the user model can be learned, which involves the learning of Bayesian networks. Empirical data gained in experiments are exploited by the learning component and the decision-theoretic planning component.)

the purpose of user modeling: (1) their ability to handle uncertainty, which is typically pervasive when making inferences about a user, and (2) their interpretability.

READY employs dynamic Bayesian networks to model user properties that change over time. In particular, these properties are time pressure (as it is felt by the user himself—in contrast to time pressure that can objectively be measured with regard to a deadline) and cognitive load. Evidence for time pressure and cognitive load can be gained from symptoms in the user's input.

A dynamic Bayesian network consists of a sequence of *time slices*. Each time slice of the dynamic Bayesian network used in the READY prototype contains three types of nodes: (1) *dynamic nodes*, such as nodes that represent time pressure or cognitive load, (2) *temporary nodes*, such as nodes that represent occurrences of disfluencies or pauses in spoken user input, and (3) *static nodes*, such as nodes that represent the base rates of occurrences of disfluencies or pauses. Figure 4.5 describes the dynamic Bayesian network as it is used in the READY prototype.

Links between dynamic nodes of consecutive time slices represent causal dependencies that persist over time. Temporary nodes have a direct influence only to nodes within the same time slice. Static nodes have only a single instantiation in the complete dynamic Bayesian network and have a direct influence on temporary nodes in all time slices.

Dynamic nodes of time slice n depend on nodes of time slice $n - 1$. For example, the assessment of cognitive load at time n depends on the assessment of cognitive load at time $n - 1$.

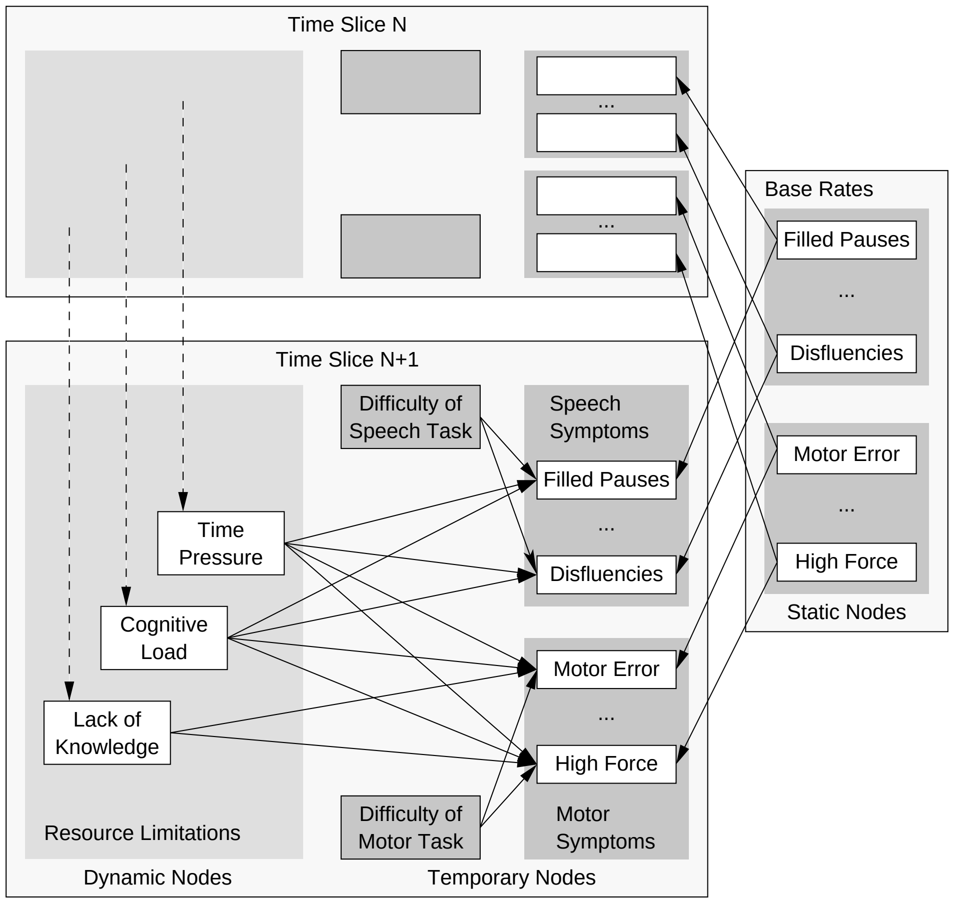


***Figure 4.5**: Dynamic Bayesian network for the recognition of a user's resource limitations.*
(A new time slice is added each time when new evidence becomes available to he system. Dynamic nodes represent properties of $\mathcal{U}$ that have a probabilistic relation to corresponding instances of the preceding time slice, temporary nodes represent symptoms in $\mathcal{U}$'s input in a particular time slice, and static nodes represent stable properties of $\mathcal{U}$'s behavior—in particular the base rates of symptoms.)

Temporary nodes are used to represent information that is only relevant at a particlar point in time. For example, the dynamic node cognitive load depends on the temporary node disfluencies. Static nodes have only a single instance in a dynamic Bayesian network. For example, the base rates for speech symptoms do usually not change during the interaction of the system with the user and can therefore be modeled as static nodes. The current assessment of speech symptoms (within a single time slice) depends on the corresponding static node that represents the base rate of the speech symptom under consideration.

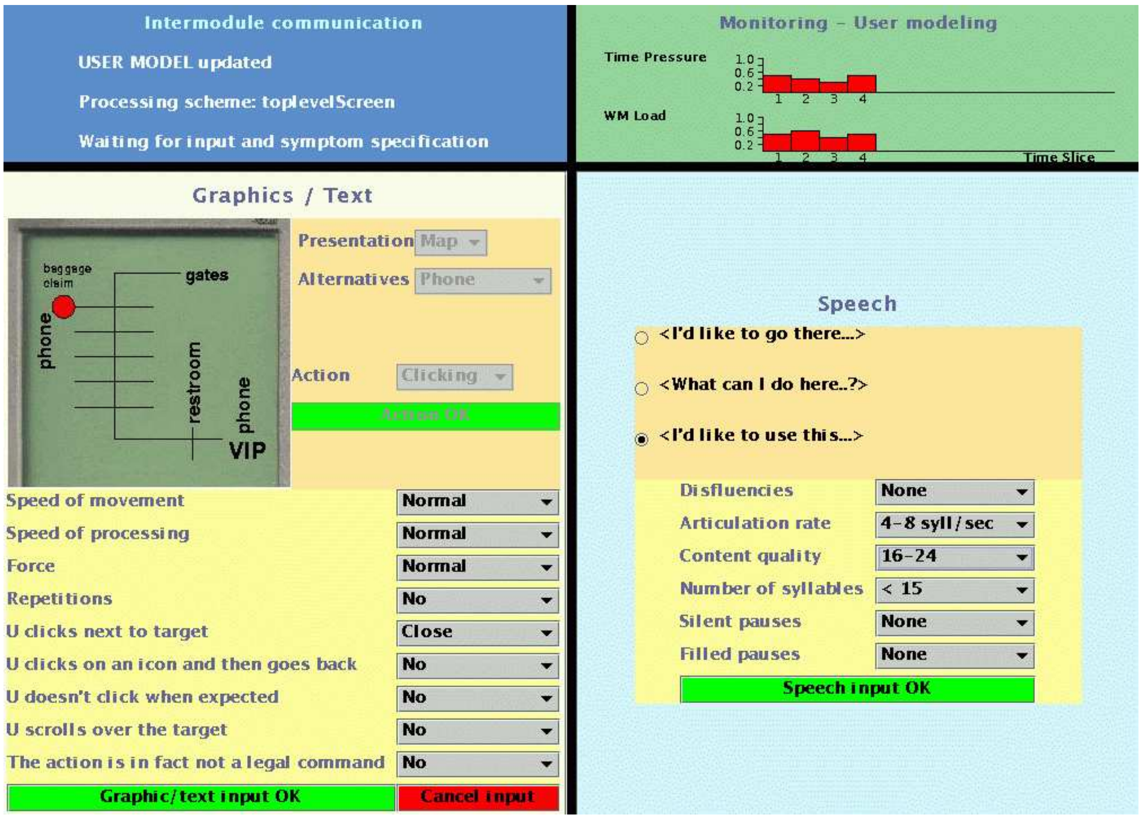
A more detailed explanation about how the READY prototype gains evidence for time pressure and cognitive load from symptoms that occur in the interaction between the system and the user are described, for example, by Jameson et al. (2000) and Lindmark (2000).

The READY User Modeling component based on Bayesian networks is only an example for a component that yields information which can be used by a decision-theoretic planner. If, for example, a user modeling component using Dempster-Shafer theory or fuzzy logic (see Section

2.1.3.7) is designed in a way that it yields similar assessments of cognitive load and time pressure, then such a user modeling component can be connected to a decision-theoretic planner in the way described in Section 4.1.6.3.

4.1.6.2 The User Interface of the READY Prototype

The user interface of the READY Prototype is realized on a workstation display as a simulation of a handheld device. To be able to simulate the use of a broad range of types of input and output, the prototype does not include components such as real speech recognizers and synthesizers; instead, all input and output is specified with the typical controls of graphical workstations (e.g., menus and graphical and textual displays).



USER MODEL updated

Processing scheme: toplevelScreen

Waiting for input and symptom specification

Presentation: Map
Alternatives: Phone
Action: Clicking

Speed of movement	Normal
Speed of processing	Normal
Force	Normal
Repetitions	No
U clicks next to target	Close
U clicks on an icon and then goes back	No
U doesn't click when expected	No
U scrolls over the target	No
The action is in fact not a legal command	No

○ <I'd like to go there...>
○ <What can I do here..?>
● <I'd like to use this...>

Disfluencies	None
Articulation rate	4-8 syll/sec
Content quality	16-24
Number of syllables	< 15
Silent pauses	None
Filled pauses	None

***Figure 4.6**: The user interface of the READY prototype.*
(The user interface is divided in four parts. Starting with the upper left part and moving clockwise, they show: (1) a trace of the inter modular communication, (2) the assessment of the user's time pressure and cognitive load, (3) the specifi cation of the speech input of the user and the speech output of the system, and (4) the specifi cation of the manual input of the user and the graphical/textual output of the system.)

Figure 4.6 shows a screen shot of the user interface of the READY prototype. The user interface is divided in four parts: (1) The upper left part, the window entitled *intermodule communication*, shows a trace of the communication between the READY components; (2) the upper right part is used to *monitor* the assessment of the user's time pressure and cognitive load; (3) the lower right part shows the simulated system's speech output and provides means to specify the user's speech input; and (4) the lower left part shows the simulated system's graphical or textual output and provides means to specify manual input (e.g., how the user has typed with the pen on an icon or how he behaved when selecting an item from a menu on the handheld computer display).

The trace of the intermodule communication is displayed merely for supervision purposes. It is used to explain the functioning of the prototype in demonstrations. The assessment of the user's time pressure and cognitive load takes place in discrete time steps. Each action of the user implies a new assessment of the user's time pressure and cognitive load. Each assessment corresponds to a new time slice in the dynamic Bayesian network that represents the user model The initial probability of time pressure and cognitive load is 0.5, that is, the system assumes that it is equally likely that the user is under time pressure or not, and that the user's cognitive load is high or not, respectively.

The specification of the user's speech input comprises information about disfluencies in the user's speech, his articulation rate, the quality of an utterance, the number of syllables of an utterance, as well as information about silent and filled pauses. The specification of the user's manual input (e.g., typing on the handheld computer display with a pen) comprises information about the speed of movement (e.g., how quick does the user scroll through a menu), the speed of processing (e.g., how long does the user need to select an item of a menu), the force of typing on the handheld computer display[4], information about repetitions (e.g., does the user click on an icon several times although only a single click is required), the information if the user clicks only close to a target (e.g., an icon) instead of exactly on the target, and other informative patterns of behavior that can occur in particular forms of manual input.

4.1.6.3 The Influence of the User Model

What impact should it have on the dialog planning process when the user modeling component has inferred a high degree of time pressure? We have said before that rewards, costs, and probabilities used in the decision-theoretic planning process provide the interface to a system's user modeling component. One way of modeling time pressure is to assign a high cost to each second of time that is required by an action performed by the system or the user. In practice, it is easier to achieve the same effect by decreasing the reward(s) for the goals state(s). Where the system is planning sequences of instructions, increasing the cost of time causes the system to derive policies that involve relatively large bundles of instructions. This is the general result that one would expect intuitively, since such policies tend to lead to faster task completion.

When the user modeling component has inferred a high degree of cognitive load of the user, the probabilities of correct executions of actions by the user should be lowered. For the scenario considered here, no empirical data for the estimation of these probabilities are available. Section 6.4 discusses various possible strategies for estimating parameters of this sort. However, the data of the experiment introduced in Section 3.1.1 allowed at least a rough estimation of the relevant transition probabilities for a low and a high level of cognitive load. If we call these probability levels $p_{i,n,low}$ and $p_{i,n,high}$ for the execution of the ith instruction of a bundle of size n, then the actual transition probability $p_{i,n}$ of the ith instruction of a bundle of size n, given an evidence for

[4]The described features are not standard for most available handheld devices, but there exist already some devices that are equipped with a sensor of this kind.

cognitive load $e_{CL} \in [0, 1]$, can be computed as

$$p_{i,n} = p_{i,n,low}(1 - e_{CL}) + p_{i,n,high}e_{CL}. \tag{4.1}$$

The policies that result if the system has high evidence for cognitive load tend to be more careful (involving smaller bundles of instructions or even stepwise presentation), while the policies that result if the system has low evidence for cognitive load, the system pushes the completion of the whole sequence of instructions giving instructions in larger bundles.

4.2 Navigation Assistance in Complex Buildings

The problem considered in the last section involved only a single goal: executing a sequence of instructions correctly. Thereby, the decision-theoretic approach allowed us to trade off achieving the goal as fast as possible versus achieving the goal as reliably as possible.

In this section, we examine a problem with two competing goals. In addition, the structure of the problem considered in this section has a topological component that will be reflected in the structure of used Markov decision process. We explain a decision-theoretic planning approach for the purpose of providing navigation assistance in a complex building. The approach copes with both navigation planning and presentation planning in a single decision-theoretic model.

There exists a range of systems that assist their users in some sort of (pedestrian) navigation task. Before we will explain our decision-theoretic approach in the context of navigation assistance, we will survey some of the systems that have called attention in the user-adaptive system and IUI community in the past. We will discuss our approach in comparison to the selected systems of this survey in Section 4.2.1.7.

4.2.1 Guides Based on Handheld Computers

While car navigation systems already approach the level of standard equipment in current models of (in particular upper class) cars, pedestrian navigation systems are still an exception in practice. However, pedestrian navigation tasks appear in the context of diverse scenarios: For example, shopping guides, tourist guides, or conference guides might involve some sort of navigation assistance. It is rather a question of acceptance than of feasibility (cf. Chapter 7) if systems for pedestrian navigation tasks make it to everyday life. This section gives a notion of the kind of applications that we possibly may take for granted within the next few years. The inclined reader may wish to consider the theses of Baus (2003) or Kray (1995) for a broader survey of the field.

4.2.1.1 Asthana et al. (1994): Personalized Shopping Assistance

An early example of an intelligent mobile system put into practice is PSA, the Personal Shopping Assistant introduced by Asthana, Cracatts, and Krzyzanowski (1994). It claims to personalize the attention provided to a customer based on individual needs without limiting his movement or causing distractions for other people in the shopping center. The PSA is mounted to the shopping cart and connected to a central server via a small area wireless network. The central server maintains a customer database (some sort of user model) and a store database (containing the information about products, product availability, etc). The system was designed to provide audio (via head set) and visual responses to inquiries of up to some hundreds of customers in real-time. The services provided by the system include guidance through the store, providing the customer with details on products of interest, pointing out items on sale, doing comparative price analysis, reminding the customer of products that were not available at the previous visit, and helping to locate an item.

The PSA is characterized by four basis functions and features: (1) simplicity of use and hands-free operation, (2) helping the customer to localize items, (3) engaging the customer by a friendly conversation and keeping his attention, and (4) helping the customer to focus to new products or products that are of particular interest for him.

The hardware as it is described in the article would allow the system at most to roughly estimate the customer's current location. The system does not seem to have the possibility to find out the customer's orientation automatically, that is, it cannot provide navigation recommendations from a user-centered perspective.

4.2.1.2 Malaka and Zipf (2000): *Deep Map*

Malaka and Zipf (2000) and Kray (1995) describe Deep Map, a mobile tourist guide prototypically implemented for the city of Heidelberg. Deep Map integrates information from different sources, such as geographical information systems (GIS), multi-media databases, and interactive Internet data sources (e.g., reservation systems). Deep Map provides guided walks that consider personal interests and needs, social and cultural backgrounds (age, gender, education), and context information (season, weather, traffic conditions, financial resources).

Deep Map is realized as a multi agent system organized in three conceptual layers: (1) The interface layer comprises all agents that interact directly with the user, (2) the cognition layer consists of agents reasoning about the user's request, and (3) the knowledge layer made up of agents providing data.

4.2.1.3 Sumi & Mase (2001): *PalmGuide*

Sumi and Mase (2001) describe the mobile conference assistant PalmGuide. It aims at facilitating communications among conference participants. The project combines mobile services with information kiosk services and online services to be assessed via the World Wide Web. Among other services, the system can recommend the user which presentation to visit next, based on the current time and the user's ratings for presentations he had attended earlier. In order to localize the conference participants, each participant wears an infrared badge. These badges emit a unique ID every four seconds, which is detected by the nearest sensor. Moreover, each of the badges has a button: By pushing the button, the participant can mark an interesting presentation when attending it. The infrared sensors are installed in the halls for oral session and at booths for poster sessions. This way, the system can track in which room a particular participant currently is. The use of infrared technology in the projects REAL and READY is inverse compared to its use in the context of the PalmGuide. While PalmGuide users carry the infrared transmitters (badges) whose signals are received by sensors in the environment, the environment in the projects READY and REAL is equipped with infrared transmitters (beacons) whose signals are received via the build-in infrared interface of the user's handheld device.

4.2.1.4 Wahlster et al. (2001): *SmartKom* (revisited)

In Section 2.2.3.1, we have reviewed the multi-modal dialog system SmartKom (Wahlster et al., 2001; Wahlster, 2002), which combines speech, gesture, and facial expression input and output.

The system supports situated understanding of possibly imprecise, ambiguous, or partial multi-modal input and the generation of coordinated, cohesive, and coherent multi-modal presentations. SmartKom merges three user interface paradigms (spoken dialogs, graphical interfaces, and gestural interactions) to achieve truly multi-modal interaction. The system employs an agent visualized as a life-like character to which the user can delegate a task. The agent elicits the specification of the delegated task in a collaborative dialog and elaborates a plan to achieve the user's intentional goal.

SmartKom-Mobile[5] integrates car and pedestrian navigation. The services currently provided by the system include guidance, tour recommendation, and location-based online booking of tickets for movie theaters. SmartKom-Mobile uses GPS for the positioning and GSM/UMTS for interactive transmission of information, for example, map data.

4.2.1.5 Pospischil et al. (2002): LoL@

Pospischil, Umlauft, and Michlmayr (2002) and Anegg, Kunczier, Michlmayr, Pospischil, and Umlauft (2002) describe Lol@ (Local Location Assistant), a mobile tourist guide prototypically implemented for the city of Vienna. The system is designed to run on advanced hardware like GPRS or UMTS mobile phones and to provide tourists with predefined or user defined tours. The use of the system is assumed to proceed in three phases: (1) In preparation of a sightseeing tour, the user retrieves sightseeing information (points of interest) from his hotel room. (2) The user is on the actual sightseeing tour. And (3) the user accesses tour information (e.g., a tour diary) after finishing the tour.

Different location methods and environmental conditions may cause considerable variations in positioning accuracy. The system considers this uncertainty by representing a location estimate rather than a guess for the exact position. This location estimate is represented by the graphical metaphor of a disc with the color and opacity fading from the center to the edge of the disc. The radius corresponds to the current level of accuracy.

Lol@ applies a hybrid routing concept combining automatic user positioning with GPS and explicit interaction with the user to elicit his position. The user can choose to receive the route recommendations indicated on a map on the mobile phone display or translated by a text-to-speech engine and read out by the system.

4.2.1.6 Wahlster et al. (2001): REAL (revisited)

In Section 2.2.2.6, we have reviewed the REAL system (Wahlster, Baus, Kray, & Kr¨uger, 2001; Butz, Baus, Kr¨uger, & Lohse, 2001; Baus et al., 2002), which features a seamless integration of different positioning technologies for a hybrid navigation system. REAL integrates two subsystems: (1) IRREAL (InfraRed REAL) for indoor navigation based on a special infrared transmitter infrastructure, and (2) ARREAL (Augmented Reality REAL) for outdoor navigation based on GPS. Being part of the Collaborative Research Program 378 on Resource-adaptive cognitive processes, adapting to both the limited technical resources (screen size, resolution, color capabilities, etc.) of mobile devices and the limited cognitive resources of the user (e.g., limitations due to unfamiliarity with the environment, a secondary task performed in parallel, or time pressure) are central research objectives in the project.

[5]There are three versions of SmartKom: SmartKom-Public, SmartKom-Mobile, and SmartKom-Home/Offi ce. They are described in detail by Wahlster et al., 2001.

The system considers resource limitation already during the process of way finding. For example, instead of recommending the user the shortest route, the system might choose a route that is slightly longer, but minimizes the number of turning points and thereby the number of recommendations needed to describe the route (and the level of attention that the user requires for the navigation task).

4.2.1.7 Synopsis

The systems described in the preceding sections all have different strengths: The strength of the Personalized Shopping Assistant is its ability to provide up-to-date-information about products on demand and at the place where it is needed to make decisions—right at the shelves of the supermarket. Deep Map provides guided tours that cover personal interests and needs, social and cultural backgrounds, and context information. Its strength is the ability to integrate information from different sources. SmartKom provides similar services as Deep Map, but puts a focus on multi-modal dialog capabilities (including speech, gesture, and mimics input and output). Moreover, it provides a seamless integration of outdoor pedestrian and car navigation. The PalmGuide focuses on facilitating communications among conference participants. Its strength is the combination of mobile services with information kiosk services and online services to be assessed via the World Wide Web. The hardware environment is similar to the one that we usually assume when applying decision-theoretic planning for navigation tasks. LoL@ and REAL attach importance to localization techniques. Both aim at adequately dealing with information about the user's current location. REAL additionally features a seamless integration of indoor and outdoor positioning.

The strength of the decision-theoretic planning approach is that it enables a system to plan several steps ahead and guide the interaction between the system and the user into a favorable direction. All systems mentioned above could in principle profit from the ability of planning ahead. In Section 7.2, we give an example of how a shopping guide can profit from decision-theoretic planning. A further strength of the decision-theoretic planning approach is the ability to address the user's desire to perform other tasks (achieve secondary goals) in addition to just getting from A to B. Finally, decision-theoretic planning can take uncertainty about the consequences of actions explicitly into account. As user-adaptive system planning problems usually involve considering a human user's (often unreliable) actions, many kinds of user-adaptive systems should be able to profit from the ability to take uncertainty about the outcome of actions into account. In the next section, we will see an example for uncertain user's actions that a system applying decision-theoretic planning can deal with.

4.2.2 Example Domain

We illustrate the application of decision-theoretic planning to provide navigation assistance in complex buildings with an airport assistance system that gives the user adaptive navigation recommendations to get to the gate and helps him to buy something in one of the stores along the way. That is, the system tries to accomplish a trade-off between two goals: (a) the primary goal, getting to the gate quickly and (b) the secondary goal, buying something along the way if possible. Reaching the primary goal is obligatory, reaching the secondary goal optional. We consider only one obligatory (primary) goal and one optional (secondary) goal in this chapter. More complex variants of this kind of problem (involving several obligatory and several optional goals) are treated in Chapter 5.

Figure 4.7 shows two passengers in different situations. It illustrates that it is reasonable to assist two users in different situations with different recommendation strategies. As in the example of Section 4.1, the system can make inferences about the user's situation, for example, from the symptoms in his speech. An intelligent system should be able to adapt both the content (which is reflected in the selected route for the particular user) of its recommendations and the way of presenting them (which can involve, e.g., a simple arrow on the display of his handheld computer, which is easy to follow, or a map with additional information about the upcoming part of the route, which might require more attention).

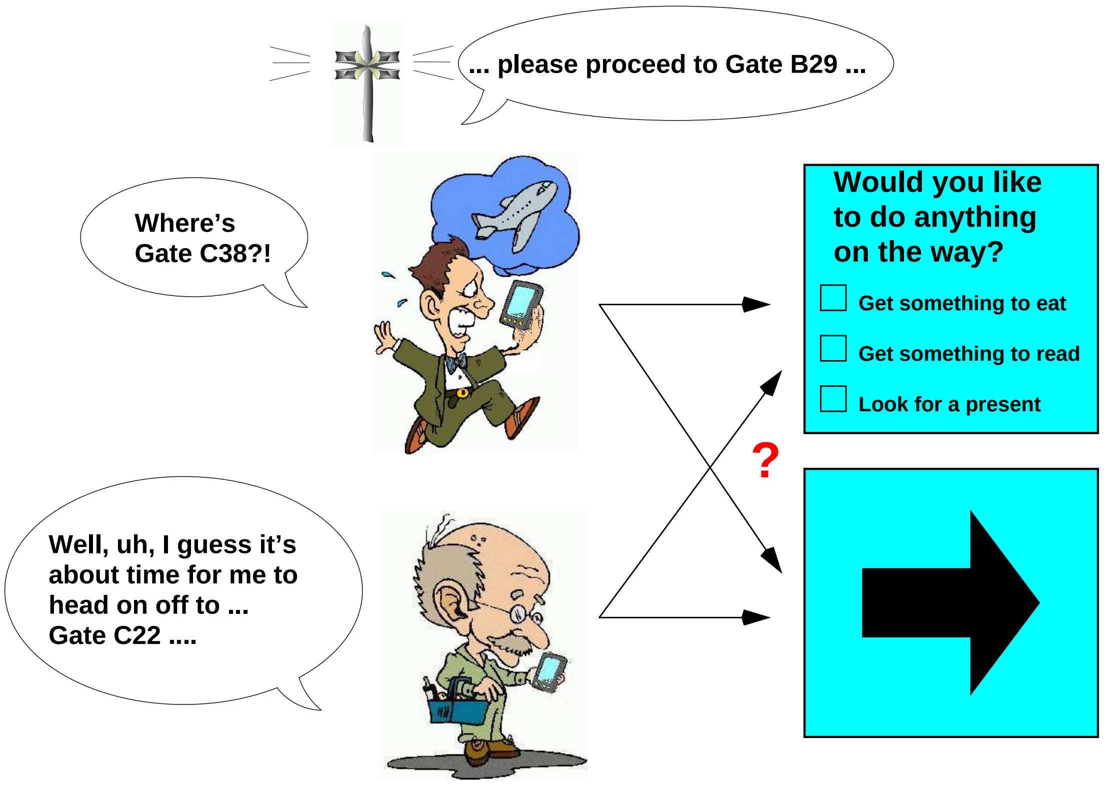

... please proceed to Gate B29 ...
Where's Gate C38?!
Would you like to do anything on the way?
☐ Get something to eat
☐ Get something to read
☐ Look for a present
?
Well, uh, I guess it's about time for me to head on off to ... Gate C22 ....

***Figure 4.7**: Navigation assistance for getting to the gate in two different situations.*
(The middle-aged man, who obviously suffers from time pressure, should rather be guided to the gate on a direct route with simple route recommendations—ignoring the desire of buying something for this wife. The senior, who seems to have plenty of time before having to reach the gate, might rather appreciate being guided along some shops, where he might like to use his spare time to buy a gift for his grandchild.)

We consider a handheld system that gives recommendations (and other information) to a visitor $\mathcal{U}$ in an airport terminal. Assuming that $\mathcal{S}$ gives $\mathcal{U}$ directions as to how to get from his current location to his departure gate, there is often significant uncertainty about what will happen when $\mathcal{U}$ has been given a recommendation. For example, $\mathcal{U}$ may choose not to follow the recommendation; and if he does follow it, his action may not bring about the intended result. Suppose $\mathcal{U}$ would like to pick up a gift on the way to the gate but would also like to get to the gate relatively soon (e.g., in order to get a good seat assignment). Then $\mathcal{S}$'s navigation recommendations should take into account both the time it will take $\mathcal{U}$ to follow the recommendations and the likelihood that $\mathcal{U}$ will see a suitable gift in one of the shops along the way.

Figure 4.8 depicts a simple environment for a scenario of this sort. There are two possible departure gates on the right, as well as a number of shops distributed all over the terminal building. In the center of the terminal, there are four regions of hight pedestrian traffic. In these areas, walking from one location to the next has a relatively high time cost. Moreover, there are two locations (Location 5 and Location 11, indicated with larger circles) with particularly high probability that $\mathcal{U}$ makes a wrong turn: If $\mathcal{U}$ is instructed to go east or west at these places, he might end up going north or south with some probability.

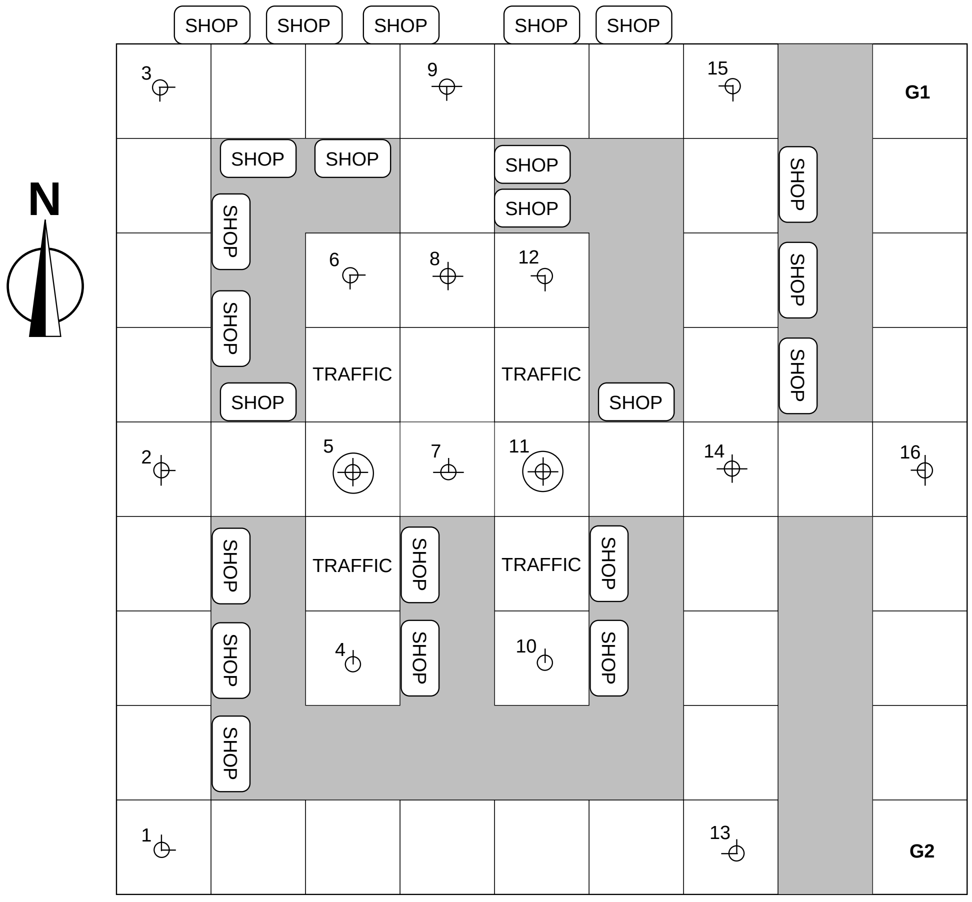


***Figure 4.8**: Topology of the airport terminal referred to in the example.*
(Each numbered circle represent a transmitter. G1 *and* G2 *are departure gates. At* Locations 5 *and* Location 11*, the probability the* $\mathcal{U}$ *takes a wrong turn is particularly high. The areas north and south of* Locations 5 *and* Location 11 *are often crowded, i.e., moving in these areas takes more time than in other corridors. A number of shops is spread all over the considered part of the terminal building.)*

For simplicity, we assume that $\mathcal{S}$ can choose between two presentation modes for its recommendations: (a) *speech mode*, in which $\mathcal{S}$ presents a recommendation in simple speech; and (b) *map mode*, which displays a map of the area around $\mathcal{U}$'s current location. Each map shows the recommended next destination, but it also offers supplementary information about the location and nature of the shops in the area. Speech mode will tend to make $\mathcal{U}$ move faster, but map mode will will increase the likelihood that $\mathcal{U}$ will find a gift. That is, by well considered application of either one or the other mode, $\mathcal{S}$ can influence the time that $\mathcal{U}$ will need to get to the gate and the likelihood that $\mathcal{U}$ will find a gift on the way to the gate.

Finally, we assume that, whenever $\mathcal{U}$ is near a transmitter, $\mathcal{S}$ is aware of both $\mathcal{U}$'s current location and orientation, as well as whether $\mathcal{U}$ has found a gift (the latter piece of information perhaps being supplied explicitly by $\mathcal{U}$). The information about $\mathcal{U}$'s current location and orientation can be gained, for example, from infrared transmitters (or by beacons repeatedly emitting their ID) that could be installed in the airport building. We will introduce a suitable hardware configuration for this purpose in the context of the user study reported in Section 7.2.

We assume the information about $\mathcal{U}$'s current location and orientation, as well as the information whether $\mathcal{U}$ has found a gift or not, to be fully observable. The technology that the scenario is based on provides reliable feedback about $\mathcal{U}$'s current location and orientation: A proper arrangement of the infrared beacons guarantees that $\mathcal{U}$'s handheld device will never receive signals from more than one beacon at a time, that is, the mapping from received signals to a location and orientation is unambiguous. $\mathcal{S}$ makes decisions and communicates recommendations only if it receives a signal of any beacon. It does not provide any assistance while $\mathcal{U}$ is between two distinct places marked by a beacon. If $\mathcal{S}$—assuming the given hardware configuration, that is, not equipped with any additional means for localization—was supposed to provide assistance between two beacons, it could only guess its current location based on the observation of the last beacon. This would require a partially observable decision-theoretic model (cf. also Section 4.1.3.2). The same holds if we do not assume that the user reliably gives feedback about having bought a gift: If we wanted $\mathcal{S}$ to consider the possibility that $\mathcal{U}$ might have bought a gift but has not indicated his purchase to the system, this would again involve a part of the decision-theoretic model to be considered as partially observable. The type of modeling described in the next section is appropriate only if there is not significant uncertainty about the consequences of actions once feedback has been obtained. The user study described in Section 7 illustrates that at least in some cases this assumption holds in our example domain.

4.2.3 Model

This section describes how the example domain can be modeled in terms of a fully observable Markov decision process. Each state in the transition model has two features: (1) the current location of $\mathcal{U}$ and the information as to whether $\mathcal{U}$ has bought a gift or not—we denote this information by a "+" or "–". This means that there are two states for each of the 18 locations (including G1 and G2) of the transmitters, which results in 36 states altogether.

Each recommendation action comprises a subsequent destination and a presentation mode. As Figure 4.9 illustrates, each action in a given state corresponds to one or more possible transitions to a following state. The cost of a given transition is the time that $\mathcal{U}$ will require to reach the corresponding subsequent destination. If $\mathcal{U}$ is in a state tagged with a "–" (i.e., he has not yet bought a gift), he will reach the adjacent location and will have bought a gift with a particular probability p, and he will reach adjacent location and will not have bought a gift with probability $1 - p$. If $\mathcal{U}$ is in a state tagged with a "+" (i.e., he has already bought a gift), then he will reach the adjacent location and will have bought a gift with certainty ($p = 1$). This merely describes that $\mathcal{U}$ will never lose a gift that he has already bought.

The cost of a transition leading from a "–" to a "+" state is much higher than the cost of a transition from a "–" to another "–" state or from a "+" to another "+" state. This difference is due to the assumption that if $\mathcal{U}$ in fact has bought a gift on the way from one location to another, then

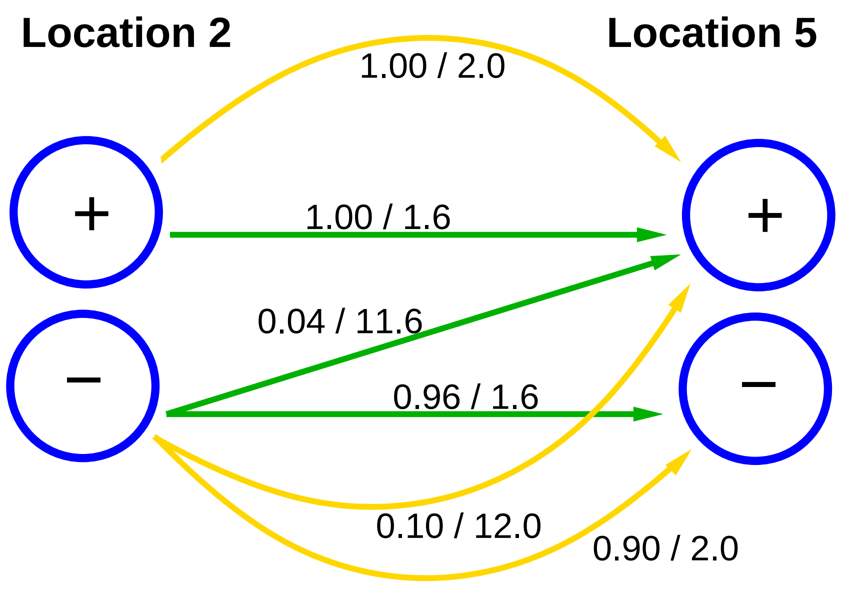


***Figure 4.9**: State transitions, probabilities, and costs associated with a recommendation that $\mathcal{U}$ go from* Location 2 *to* Location 5.
(Dark and light arrows represent possible transitions when $\mathcal{S}$ uses speech mode and map mode, respectively. Each pair of numbers next to the arrow gives (a) the probability that the corresponding transition will be made if the recommendation is given; and (b) the cost of making that transition.)

he has entered one or several shops, has looked for a gift for a while, has found something, and had to stand in line to pay for the item. If he does not buy a gift on the way from one location to another (either because he has already bought a gift or none of the shops on the current part of the route seem interesting to him—this may also be because $\mathcal{S}$ does not make him aware of the shops along the way), he either does not enter shops at all or he enters a shop, realizes rather quickly that he will not find a gift in this shop, and moves on. The model is not appropriate for a user who tends to frequently enter shops, spend a rather long period of time there, and then leaves again without having bought anything. However, we could construct a different model that captures the characteristic behavior of such a user.

Uncertainty about $\mathcal{U}$ going into the recommended direction can be handled similar to the uncertainty about $\mathcal{U}$ finding a gift on his way from one location to another. Figure 4.10 describes the relevant part of the decision-theoretic model for the case that $\mathcal{U}$ has already bought a gift and is recommended to go from Location 11 to Location 14. Here, the assumption is that a recommendation in map mode will rather increase the probability (compared to speech mode) that $\mathcal{U}$ will take a wrong turn and end up in Location 10 or Location 12. Note that this assumption is not empirically validated. However, if it turned out that this assumption does not hold (e.g., that it would be the other way round, i.e., speech mode would increase the probability that $\mathcal{U}$ will take a wrong turn) then the model can be corrected by simply adjusting the probabilities. Note also that we have assigned a relatively high cost to (no matter if accidentally or intendedly) go from Location 11 to Location 10 or Location 12 to indicate that these corridors are crowded. This assignment illustrates that the costs of a transition in this scenario does not correspond to the physical distance between two locations, but rather to the time needed to go from one location to another.

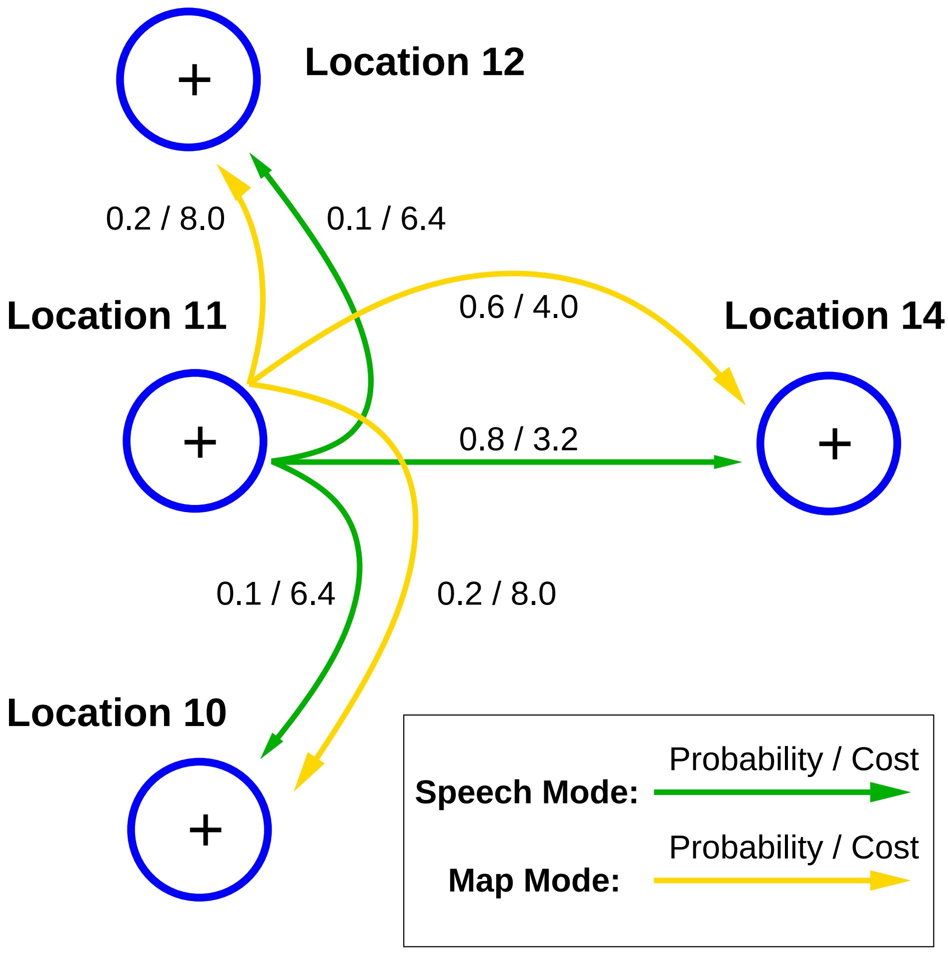


***Figure 4.10**: State transitions, probabilities, and costs associated with a recommendation that $\mathcal{U}$ go from* Location 11 *to* Location 14 *if he has already bought a gift.*
(Dark and light arrows represent possible transitions when $\mathcal{S}$ uses speech mode and map mode, respectively. Each pair of numbers next to the arrow gives (a) the probability that the corresponding transition will be made if the recommendation is given; and (b) the cost of making that transition.)

There are also two states for each of Location G1 and Location G2, the gates. The reward for reaching the gate without a gift (the state at G1 or G2 tagged with a "–") is 0; and the reward for reaching the gate with a gift (the state at G1 or G2 tagged with a "+") is some nonnegative number that reflects the importance (or the additional value) that $\mathcal{U}$ attaches to the goal of finding a gift, relative to the goal of arriving as early as possible at the gate[6]

The reward and the time costs together determine the expected utility of any possible trip of $\mathcal{U}$ to the gate. $\mathcal{S}$'s job is to compute the policy—the mapping from states into recommendation actions—that has the highest expected utility. Such a policy is sensitive to the probabilities of transitions, the costs of transitions, and the rewards of goal states. These sensitivities jointly allow to express (1) $\mathcal{U}$'s desire to find a gift, that is, the importance that $\mathcal{U}$ assigns to finding a gift on

[6]To take into account, for example, the possibility that $\mathcal{U}$ will miss his plane if he arrives after a certain point in time, a more complex modeling of time, such as it is described in Section 4.4, is required.

the way to the gate relative to getting to the gate early and (2) $\mathcal{U}$'s reliability to follow navigation recommendations correctly. The following section will explain some examples to illustrate these relations.

4.2.4 System Behavior

Figure 4.11 and 4.12 show the policies that were computed for eight situations that differ along two dimensions: $\mathcal{U}$'s "reliability"—that is, his ability to avoid making wrong turns—and the importance to $\mathcal{U}$ of finding a gift. We will explain the figures top down.

The two graphs in the top row of Figure 4.11 show the recommendations that $\mathcal{S}$ will make in the case where $\mathcal{U}$ attaches absolutely no importance to a gift—or where he has already found a gift. The reliable $\mathcal{U}$ is steered towards the gate as quickly as possible, even when this means that he will pass through one of the locations where other users might make a wrong turn. The less reliable $\mathcal{U}$, in contrast, is drawn away from the central area of the terminal whenever this is possible because of the danger that he might make a wrong turn and waste time in one of the congested areas. For both users, all of the recommendations are presented in speech mode; the slower map mode can be worthwhile only if there is some value to finding a gift.

The graphs in the second row of Figure 4.11 show parts of policies that are suitable when $\mathcal{U}$ attaches some (low) importance to finding a gift. As soon as $\mathcal{U}$ has found a gift, $\mathcal{S}$ will switch to the corresponding recommendations shown in the top two graphs; for clarity, these recommendations are not included in the other graphs. The reliable $\mathcal{U}$ is still directed to the gate along the fastest possible route, that is, the policies in the left column are the same because $\mathcal{S}$ does not expect the reliable $\mathcal{U}$ to get lost in the crowded areas of the building. The unreliable $\mathcal{U}$ is still steered away from the dangerous locations. But an interesting change of system behavior is reflected in the policy for the unreliable $\mathcal{U}$ at the middle location in the left-hand part of the building: Instead of directing $\mathcal{U}$ northward, through the area with the most stores, $\mathcal{S}$ sends him southward. The reason is that, if $\mathcal{U}$ passed through the store-rich area, he might stop to buy a gift, although according to his expressed preferences it would not be worthwhile for him to do so

If $\mathcal{U}$ has expressed moderate interest in a gift (top row of Figure 4.12), he is given some recommendations in map mode (lighter arrows). From some locations, both users are led to the region in the upper left-hand corner with the highest shop density. Here, $\mathcal{S}$ will advise $\mathcal{U}$ to go back and forth, past the shops, until he finds a gift, at which point $\mathcal{S}$ will lead him to the gate in the way shown in the two graphs in the top row of Figure 4.11. According to the probabilities specified in the model, $\mathcal{U}$ is likely to find a gift quite quickly in this area, such that there is no danger of a long or infinite loop.

Finally, if $\mathcal{U}$ has expressed a sufficiently strong interest in finding a gift (bottom row of Figure 4.12), $\mathcal{S}$ will direct him towards the area with the highest density of shops, even when he is already very close to his destination gate G1. In this situation, the time costs of walking around the airport are dominated by the perceived benefits of finding a gift.

Note that these eight examples are only selected snapshots of the policy space. They cover the system behavior only for some of the most interesting situations. Probabilities and rewards are continuous variables, and each slight modification can in principle result in a slight modification of the recommendation policy. Appendix D lists a part of a sample trace of the computations for an instance of problem considered in this section.

The decision-theoretic approach does not necessarily need to cover all aspects of the process to be planned, that we consider in this section. For example, the decision-theoretic model can

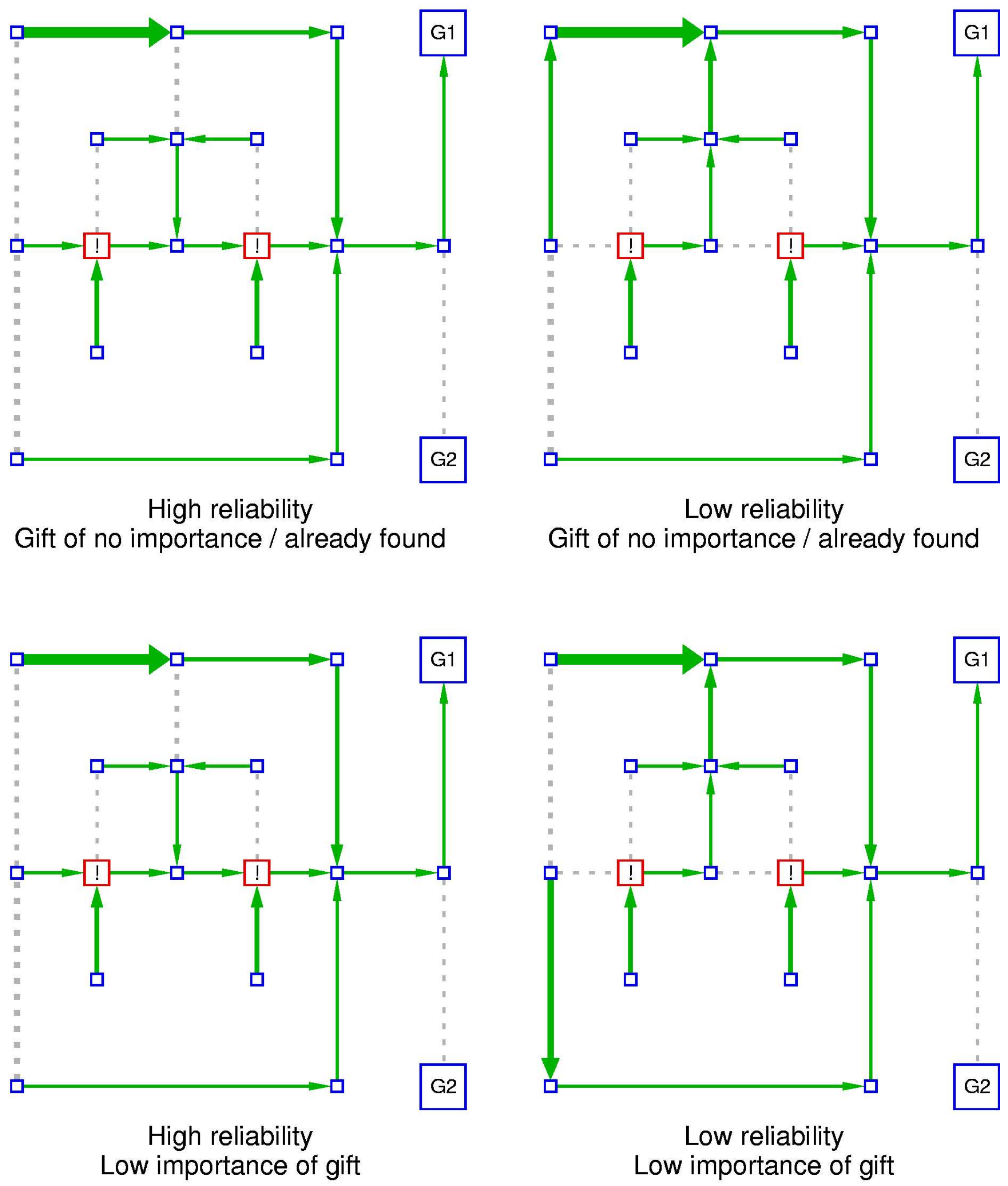


***Figure 4.11**: Policies as a function of $\mathcal{U}$'s ability to follow recommendations correctly and $\mathcal{U}$'s desire to find a gift.*

(Dark and light arrows represent recommendations given in speech mode and map mode, respectively. Dashed lines show where movements are possible but not recommended. The width of an arrow or a line between two locations reflects the number of shops between them. Each graph describes only a part of a policy, for example, the two graphs in the top row describe the part for the situation in which $\mathcal{U}$ has already found a gift (or the gift is assigned no importance), the two graphs in the bottom row describe the part for the situation in which $\mathcal{U}$ has not yet found a gift and the gift is assigned low importance.)

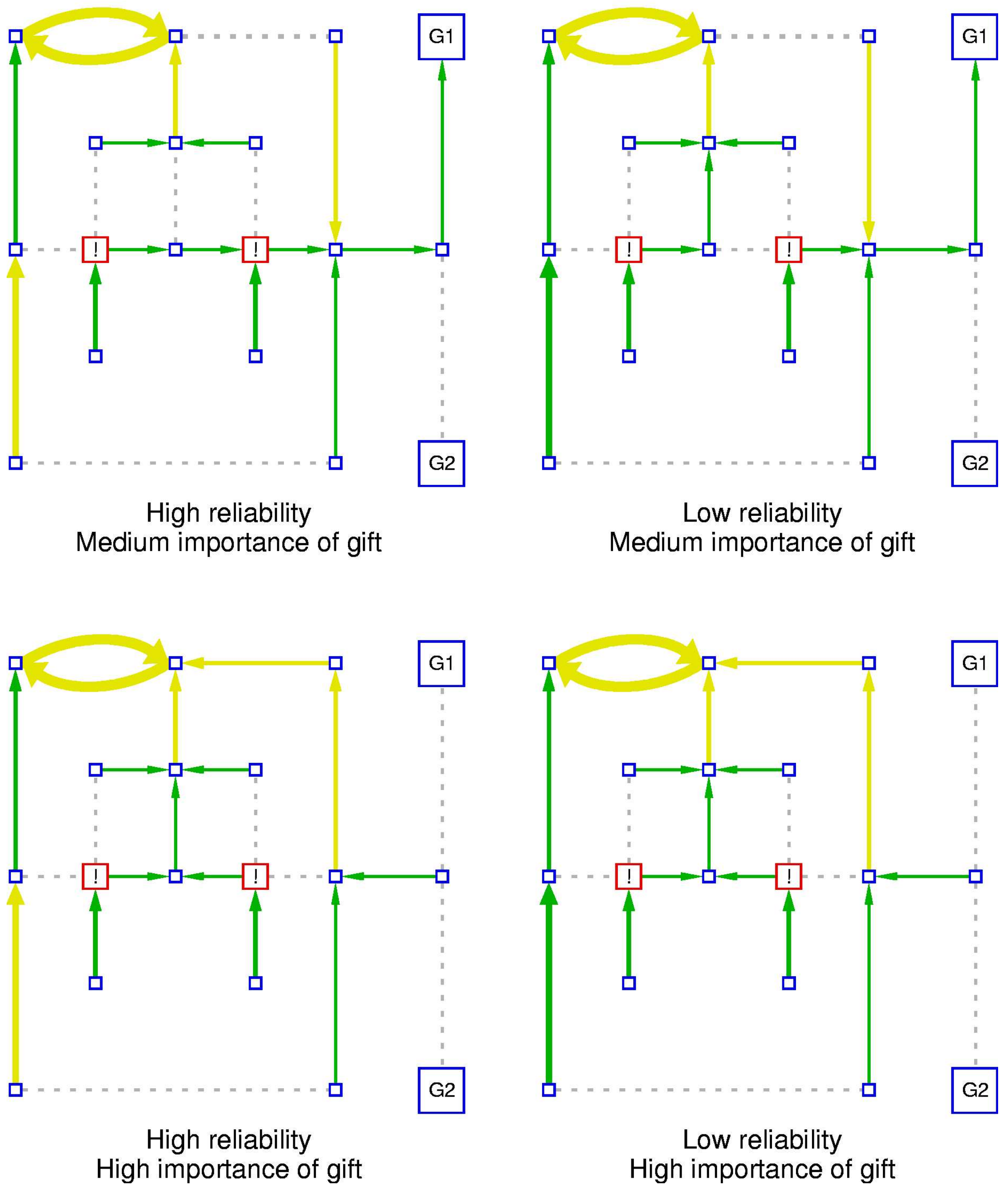


***Figure 4.12**: Policies as a function of $\mathcal{U}$'s ability to follow recommendations correctly and $\mathcal{U}$'s desire to find a gift.*

(Dark and light arrows represent recommendations given in speech mode and map mode, respectively. Dashed lines show where movements are possible but not recommended. The width of an arrow or a line between two locations reflects the number of shops between them. Each graph describes only a part of a policy. All graphs of this figure describe a part of a policy for the situation in which $\mathcal{U}$ has not yet found a gift.)

be modified such that it covers only the *navigation planning* (i.e., the selection of the direction) or only the presentation planning (i.e., the selection of the presentation mode). However, it is a strength of the introduced decision-theoretic model that it can consider both aspects of the process to be planned in an integrated manner: The selection of a particular direction at one location can influence the presentation mode at another location and vice versa. Both, the recommended directions and the selected presentation mode at a particular location have an effect on the expected utility of a recommendation. Therefore, it is reasonable to optimize the expected utility of a sequence of recommendations considering both aspects of each recommendation. Section 4.3 illustrates a scenario in which it is reasonable to restrict the decision-theoretic model in a way that is covers only a part of the process to be planned.

4.2.5 Sensitivity Analysis

So far, we made the assumption that we can acquire the probabilities required for the decision-theoretic model exactly. This assumption is correct if the probabilities can, for example, be determined in a controlled experiment as the one that we have described in Section 3.1.1. We will see in Section 6.4 that there exist various approaches to determine more or less exact probabilities. However, what if the probabilities that we use to describe the process which we want to plan for are not 100% accurate? By simulation, we analyzed the quality of policies computed on the basis of inaccurate probabilities and compared them to the quality of plans computed on the basis of inaccurate probabilities for the same domain.

In preparation for the analysis, we first compute a set of plans[7] (i.e., in the scenario at hand, these are fixed sequences of navigation recommendations to guide $\mathcal{U}$ from a given location to the gate), such that from every location in the airport building, there is a plan to get to the departure gate taking the desire to buy something along the way into account. Then, we simulate both the application of the policy and of the set of plans in 1000 randomized runs for the problem at hand. That is, during each application of a policy or a plan, the event of finding a gift in any of the shops along the way occurs randomly according to the probabilities specified for the simulation of the real world process. Moreover, in each run, the gained reward is determined for each possible starting position of $\mathcal{U}$. While a policy is valid for each possible starting position anyway, from the set of plan, the right plan for the given starting position has to be selected for this purpose. The results can be seen in Figure 4.13. A sample traces of the corresponding computations is listed in Appendix E.

If the used probabilities are accurate, that is, the probabilities applied in the decision-theoretic model (and for the computation of the plans, respectively) equal the probabilities applied in the simulation, the expected utility of a policy equals (up to some positions after decimal point) the actual average reward gained in the simulations. Figure 4.13 shows that the policy is more robust with regard to underestimation (i.e., a situation where the actual probabilities are smaller than the probabilities used in the decision-theoretic model) than to overestimation. In the given example domain, this effect can be explained by an analysis of the following system behavior: If there is a high probability of finding a gift on a particular part of the way, the systems tends to send $\mathcal{U}$ along that part of the way a second or third time if he does not find a gift there at first go. This system behavior is not effective if the probabilities of finding a gift there are overestimated. Apart from the situation of heavy probability overestimation, the policy for the given example problem

[7]We did not use a sophisticated planning algorithm for this purpose but constructed the plans in a brute force combinatorial way.

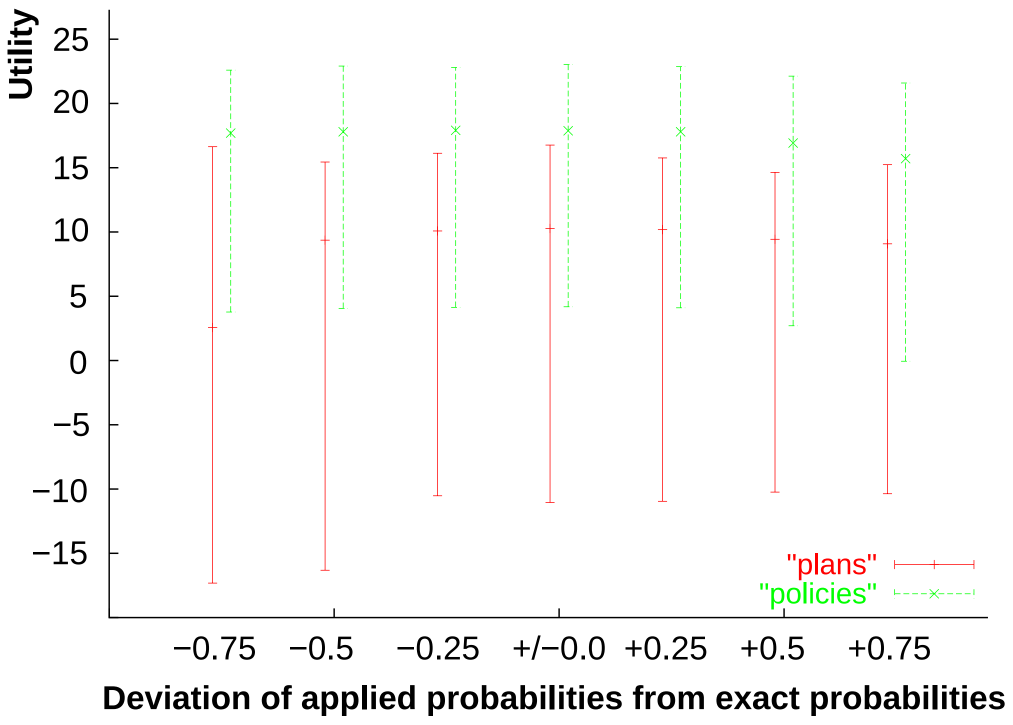


***Figure 4.13**: Average rewards for $\mathcal{U}$ following plans and policies based on more or less inaccurate probabilities in the navigation assistance domain.*
(Comparison of the average rewards gained by following plans and policies with accurate probabilities for finding a gift in the stores along the way to the gate and applied probabilities deviating from real probabilities for 25, 50, and 75 percent, thereby considering both too optimistic and too pessimistic expectations, that is, over- and underestimation of the probabilities.)

is very robust with regard to inaccurate probabilities, and even in the case of heavy probability overestimation, the policy is still useful and yields good results.

The average reward gained when applying plans is clearly less than the average reward when applying a policy. This holds regardless of the accuracy of the probabilities used in the planning process. In the example problem at hand, plans tend to be less robust with regard to heavily underestimated probabilities. The reason is that plans tend not to consider achieving the secondary goal at all if the probabilities say that the chances are low, while policies still try to achieve the secondary goal at least in some cases.

An important disadvantage of a plan in this context is that it might accept a deviation from the direct route to the gate in order to increase the probability of finding a gift although $\mathcal{U}$ has already bought a gift. However, the idea of plans is that they are applied in a *non-observable* scenario, that is, in a scenario in which it is not possible to get any feedback about the state of the process. The non-observability assumption causes plans to be completely inflexible. Policies, in contrast, are applied on the assumption that the process which they control is fully observable and therefore are much more flexible than plans.

4.2.6 Navigation Assistance in the READY Prototype

The decision-theoretic model for the airport navigation assistance domain of low complexity has been implemented for demonstration purposes. The implementation allows to visualize the system behavior for simple navigation tasks (achieving a primary goal without considering any secondary goal) and navigation tasks with one secondary goal (e.g., getting to the gate and picking up a snack from a fast food restaurant on the way). The goal selection window on the left hand side at the top of Figure 4.14 shows the combinations of a primary and secondary goals that can be dealt with in the implementation.

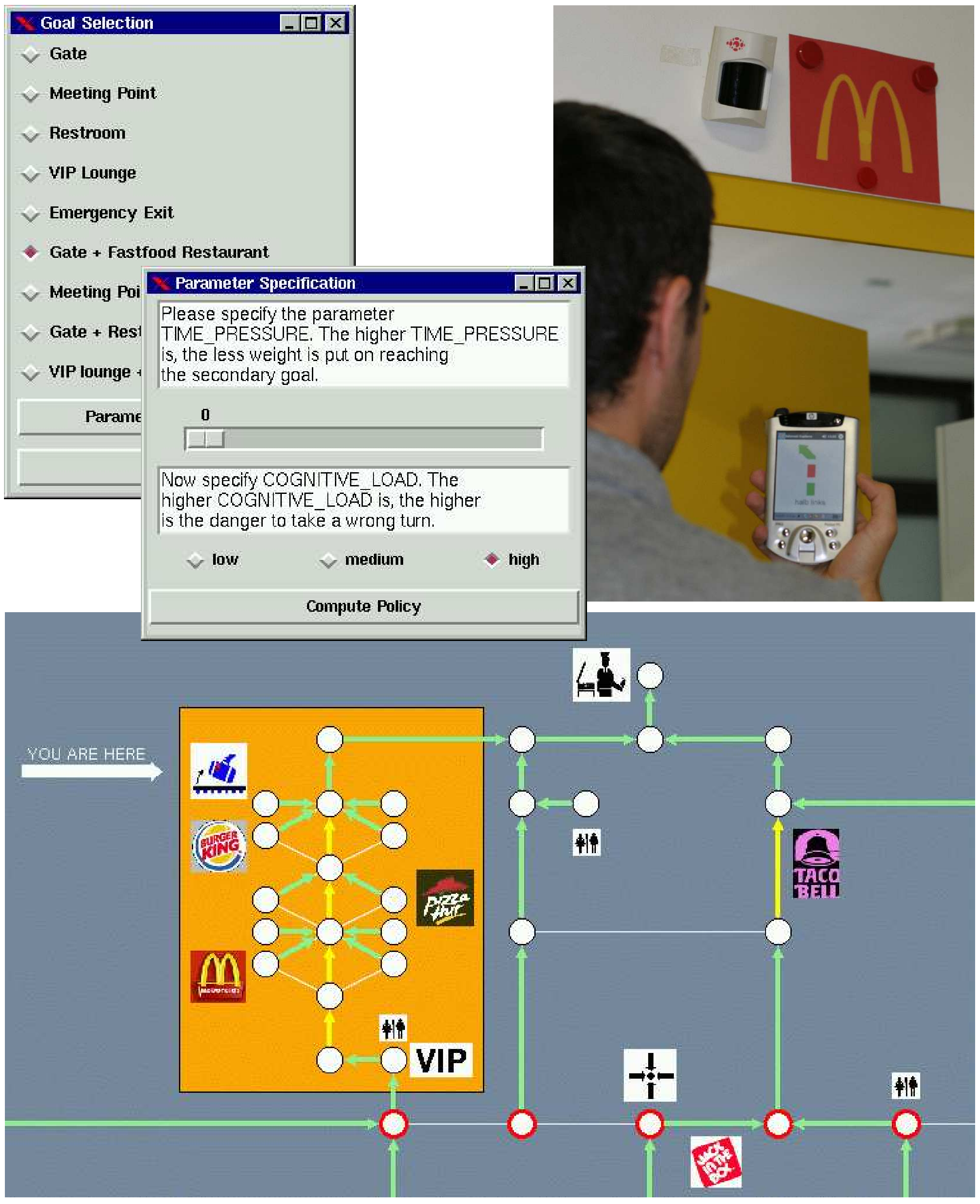
If a combination of a primary and a secondary task is selected, the parameters time pressure and cognitive load can be specified manually. The parameter specification window in the middle of Figure 4.14 allows to adjust the time pressure as it is subjectively felt by the user on a scale from 0 to 20 and to specify the cognitive load of the user as low, medium, or high. The manual specification of time pressure and cognitive load can be replaced by a user modeling component, such as it is described in Section 4.1.6.1, with little effort.

Figure 4.14 (bottom) shows a part of a screen shot of the policy as it is displayed for a user who does not feel any time pressure but whose cognitive load is high. The topology that is used in the implementation is only slightly more complex than the topology introduced in Figure 4.8. It includes a part of the physical environment of the AI lab at Saarland University—this is the highlighted part in Figure 4.14. The physical environment of the AI lab served as a mock-up of a hallway in an airport terminal hosting the baggage claim and several fast food restaurants. The decision-theoretic planner has an interface to the physical environment, such that the user can receive navigation information via the infrared transmitters mounted on the ceiling in the AI lab. If a recommendation is given in map mode, the map includes information about fast food restaurants placed in the mock-up of the airport terminal along the way. The integrated system has been demonstrated at the AI lab (see Figure 4.14, top right).

In the policy visualization, five locations are highlighted with red circles. These locations are assumed to be particularly dangerous with regard to not following navigation recommendations correctly. This assumption is reflected in the decision-theoretic model as described in Section 4.2.3 (Figure 4.10). As a result, the prototype tries to avoid these locations in particular if the cognitive load of the user is specified to be high. However, as with the decision whether to guide the user along shops or not, the decision whether to avoid a “dangerous” place or not is always based on a trade-off between getting to the gate quickly and getting to the gate without taking a wrong turn. That is, if at the same time the importance of getting to the gate quickly is high enough (which is reflected in the user’s subjectively felt time pressure), the system can decide to recommend a shorter path despite the danger that the user might take a wrong turn.

4.3 Combining Predefined Routes with Decision-Theoretic Planning

The route that a navigation system offers to a user can be planned according to miscellaneous criteria. For example, it can be the shortest or the most enjoyable path to a goal, the path with the least number of divergencies, or the path most often used by the user before. For whatever reason, the system can sometimes rely on *predefined routes*, for example, if the system is aware of the routes that the user has already taken. Under these circumstances, no planning is needed and the route descriptions can be retrieved directly from a database. These route descriptions can be used to explain the way to the user in advance (e.g., at an information kiosk), but they are not suitable



Goal Selection

- Gate
- Meeting Point
- Restroom
- VIP Lounge
- Emergency Exit
- [✓ Gate + Fastfood Restaurant]
- Meeting Poi
- Gate + Rest
- VIP lounge +

Parameter Specification

Please specify the parameter TIME_PRESSURE. The higher TIME_PRESSURE is, the less weight is put on reaching the secondary goal.

0

Now specify COGNITIVE_LOAD. The higher COGNITIVE_LOAD is, the higher is the danger to take a wrong turn.

low medium [✓ high]

Compute Policy

Figure 4.14: *Goal selection window, parameter specification window, snapshot of the system in use, and part of the recommendation policy for a user who feels no time pressure but whose cognitive load is high.*
(Top left: A combination of an obligatory primary and an optional secondary goal is selected; evidence for subjectively felt time pressure is manually set to 0, evidence for cognitive load is set to high. Bottom: Dark and light arrows represent recommendations given in speech mode and map mode, respectively. The topology used in the prototypical implementation is different from the topology of Figures 4.8, 4.11, and 4.12—but the prototype uses the model described in Figure 4.9 and 4.10. Top right: Part of the prototype was realized on a handheld computer and navigating through the airport building was simulated in a mock-up at the AI lab.)

to handle unpredicted user actions during the navigation. Predefined routes can also be augmented with additional navigation recommendations based on decision-theoretic planning to guide a user who has taken a wrong turn back to the original route. And finally, decision-theoretic planning can be used to determine the form of output presentation of the directions given by a predefined route.

4.3.1 Example Domain

In this section, we will explain how to integrate predefined routes in the decision-theoretic planning approach. We refer to the example domain introduced in Section 4.2.2. Suppose $\mathcal{U}$ has intimated that he would like to be guided to gate G1 and to buy a gift (something not further specified, which he wants to bring home for his wife) on the way to the gate. Suppose further, that $\mathcal{S}$ has already selected the route from $\mathcal{U}$'s current position (Location 1) to his departure gate G1 indicated with the dotted arrow in Figure 4.15. The reason for this selection might have been, for example, that this is the path $\mathcal{U}$ has used most often before or that the shops along this way match best $\mathcal{U}$'s shopping interests according to the user model $\mathcal{S}$ maintains of $\mathcal{U}$. We assume that, as long as $\mathcal{U}$ is on the given route, $\mathcal{S}$ should recommend to follow this route. This is a plausible assumption, for example, if it is known that $\mathcal{U}$ does not like to use moving stairways with his luggage cart, such that $\mathcal{S}$ holds a predefined route avoiding moving stairways ready for this particular user. An alternative to this assumption is, for example, that the system recommends deviations from the given route when there is a particularly good reason, such as a bargain nearby.

A predefined route for the problem at hand specifies the direction that $\mathcal{S}$ recommends to $\mathcal{U}$ only for some locations in the airport terminal. According to Figure 4.15, $\mathcal{S}$ will first recommend $\mathcal{U}$ to go from Location 1 to Location 2, then to continue to go straight ahead from Location 2 to Location 3, then to turn right and go from Location 3 to Location 9, and so forth—as it is indicated in the figure. It is not yet specified (1) how $\mathcal{S}$ should give the recommendations (speech mode or map mode) and (2) what $\mathcal{S}$ should recommend if $\mathcal{U}$ takes a wrong turn and gets to one of the locations lying outside the predefined route.

4.3.2 Model

The model that we describe in this section focuses on the determination of the presentation mode for each navigation recommendation and the system behavior if $\mathcal{U}$ deviates from the predefined route. We represent the problem in analogy to the model described in Section 4.2.3. Each location is represented by two states: One state is used to compute a recommendation for $\mathcal{U}$ when he has already bought a gift, the other when he has not yet bought a gift. The transitions between the states are annotated with probabilities and costs as before. We assume that the probability to enter a shop and to find a gift when going from one transmitter location to the next depends on the number of shops on this part of the route; the cost of a transition corresponds to the time to get from one transmitter location to the next. As before, we assume that a recommendation given in map mode increases the probability that $\mathcal{U}$ will buy a gift on the recommended part of the route, as it makes $\mathcal{U}$ aware of shops. On the other hand, a recommendation given in speech mode is short and easier to follow, such that we assume that $\mathcal{U}$ makes the upcoming part of the route a little quicker.

The key idea of the hybrid approach is as follows: As is customary, we compute the optimal recommendation (direction plus mode) for each state by our preferred standard algorithm—value

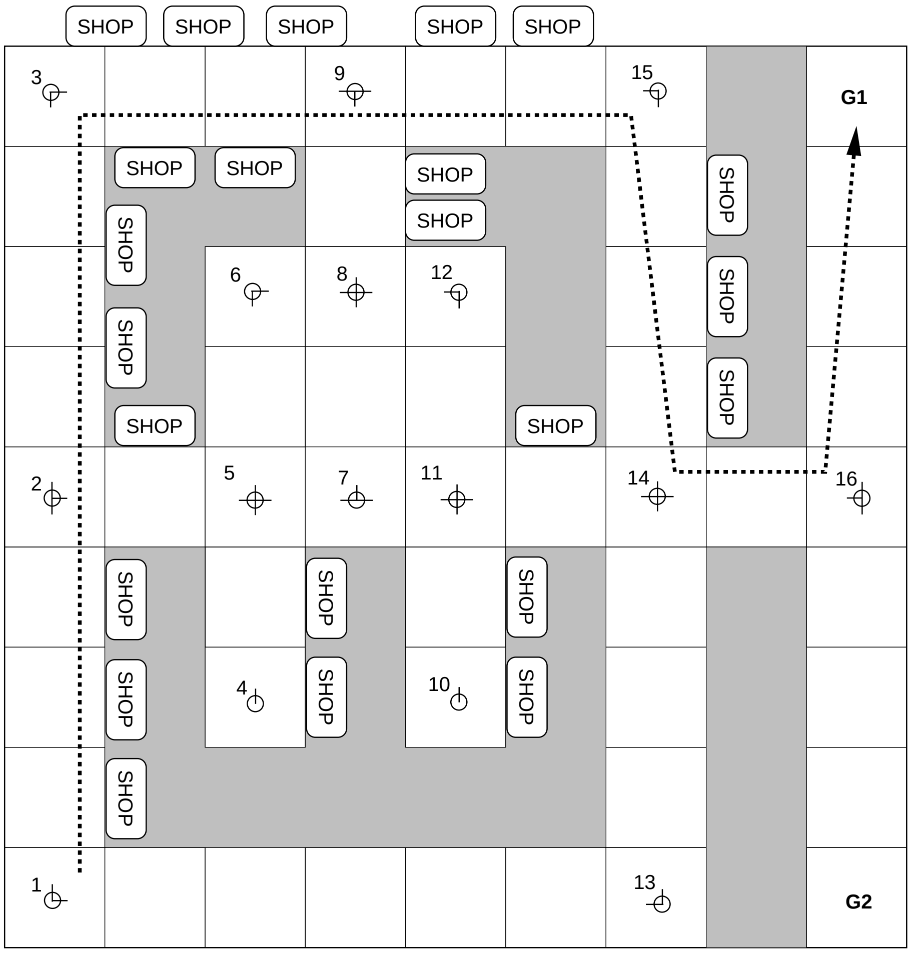


***Figure 4.15**: Topology of the airport terminal including a predefi ned path to the gate.*
(Each numbered circle represents a transmitter. G1 *and* G2 *are departure gates. A number of shops is spread all over the considered part of the terminal building. The indicated path might, for example, be the path most often used by $\mathcal{U}$ before.)*

iteration. That is, for each state of the MDP, the expected utility of each alternative recommendation is computed iteratively according to the expected utilities of the adjacent states, the probabilities each of them is reached with and the costs to reach them. The only peculiarity is that recommended directions along the predefined route are fixed, that is, we assume that there are no alternatives for directions in those states which belong to the transmitter locations on the predefined route. This approach makes sure that correct expected utilities with respect to the constraints given by the predefined route are computed for all states. It is possible to take into account how important it is to follow the predefined route. This can be achieved by adjusting the transition costs between two states of which one lies outside the predefined route. Assigning increased costs for these transitions will make $\mathcal{S}$ find routes drifting away from the predefined route less attractive than the predefined route itself. Such an assignment will consequently make $\mathcal{S}$ recommend $\mathcal{U}$ to go back to the predefined route on the shortest path.

4.3.3 System Behavior

Figure 4.16 shows parts of four generated policies that integrate the predefined route indicated in Figure 4.15. As in Section 4.2.4, the policies have been computed as a function of $\mathcal{U}$'s desire to find a gift. Additionally, the importance to follow the predefined route as good as possible is taken into account. Dark and light arrows represent recommendations in speech mode and map mode, respectively. Dashed lines show where movements are possible but not recommended. The width of an arrow or a line reflects the number of shops between two locations. We will explain the figure top down from left to right.

The two graphs in the first row describe the system behavior for a passenger who has indicated that he either has already found a gift or that finding a gift is not important to him. In both situations (low and high importance to follow the predefined route), $\mathcal{S}$ gives recommendations exclusively in speech mode: There is no need to make $\mathcal{U}$ aware of nearby shops. If $\mathcal{S}$ can assume that it is very important for $\mathcal{U}$ to follow the predefined route (lefthand side of the figure), then $\mathcal{S}$ tries to guide $\mathcal{U}$ back to the predefined route after $\mathcal{U}$ has taken a wrong turn (e.g., at Location 2, Location 9, or Location 14—see Figure 4.15). If $\mathcal{S}$ assumes that sticking to the predefined route is not as important for $\mathcal{U}$ (righthand side of the figure), then $\mathcal{S}$ tends to guide $\mathcal{U}$ on a shorter path through the center of the terminal to his gate. For example, at Location 7, $\mathcal{S}$ has three options with equal costs to guide $\mathcal{U}$ back to the predefined route. Here, $\mathcal{S}$ selects the path that leads $\mathcal{U}$ back to the predefined route at the spot which is closest to the gate (i.e., to Location 14).

The two graphs in the second row describe the system behavior for a passenger who has indicated medium importance to find a gift on the way to the gate. In both situations, $\mathcal{S}$ still gives some recommendations in speech mode, but it gives especially those leading $\mathcal{U}$ along many shops in map mode. With increasing importance, $\mathcal{S}$ would increasingly give recommendations in map mode.

Comparing the graphs column by column (upper with lower left side and upper with lower right side) yields that the directions in which $\mathcal{U}$ is recommended to go to do not change with decreasing importance of finding a gift. The property that $\mathcal{U}$ is always led back to the predefined route (in the left column) does not change, but on appropriate parts of the route, the mode in which $\mathcal{S}$ gives the navigation recommendations to $\mathcal{U}$ is adjusted.

4.4 Dealing with Fixed Deadlines

In the preceding sections of this chapter, we assumed that the interaction between a system and its user will always terminate at some point, but we do not consider this as a *fixed deadline*. Instead, our model is based on the assumption that the interaction gets more expensive the longer it takes. We can call this a *soft deadline*.

Not all deadlines in the context of human computer interaction are soft. The key idea is: A fixed deadline corresponds to an event that represents high costs for the user, which cannot be traded off against any of his options available in the scenario at hand. Consider, for example, a shopping mall scenario. Here, not finishing the shopping of all items that one needs until the closing time of the mall means having to buy the remaining items in some other shop (where they might be more expensive) or having to come back the other day to finish the shopping (i.e., $\mathcal{U}$ has costs in terms of the time needed to come back, the cost of driving to the mall a second time, or maybe the cost of not having something to eat at night). In the case of the shopping mall, the

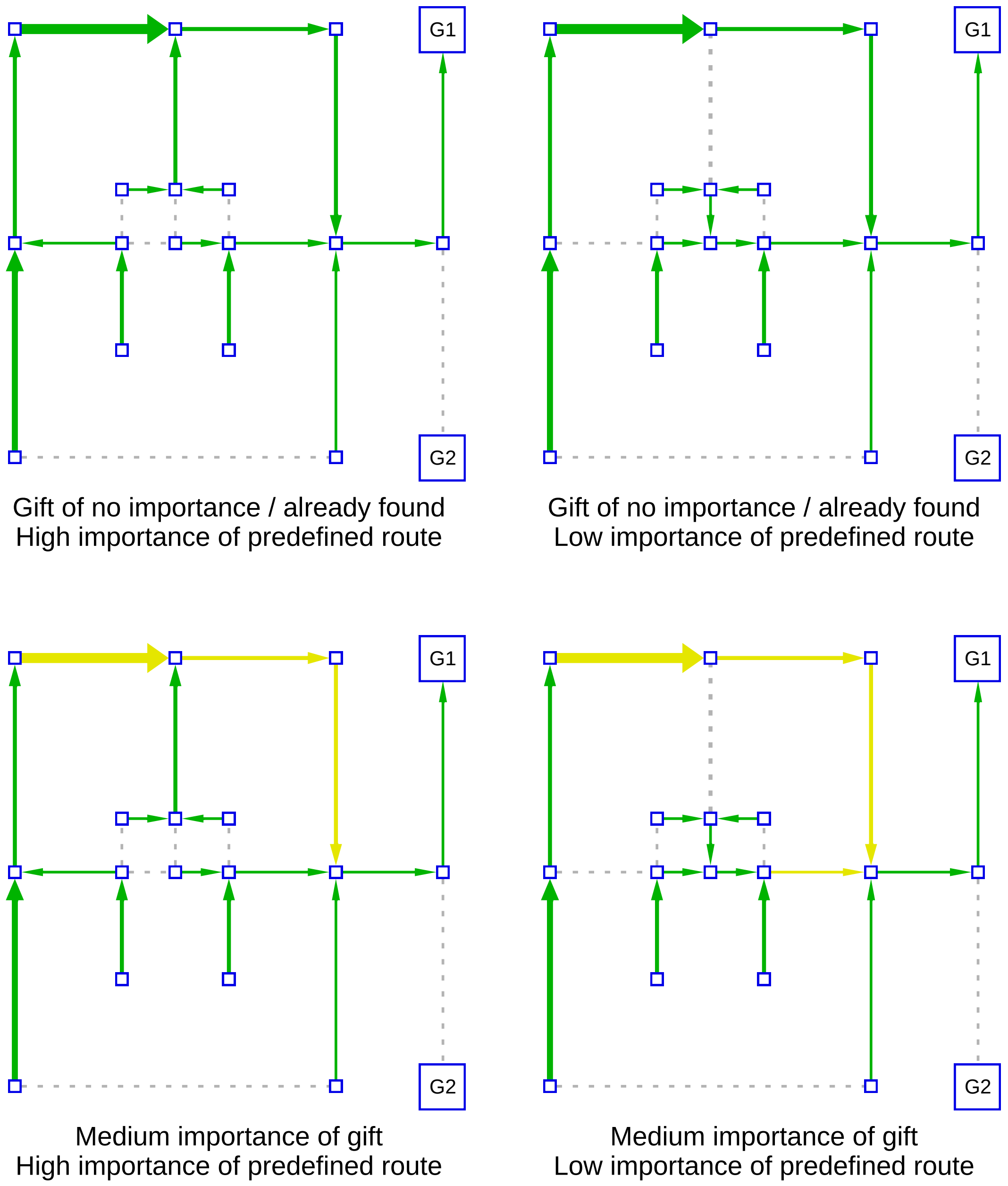

G1
G2
Gift of no importance / already found
High importance of predefined route
G1
G2
Gift of no importance / already found
Low importance of predefined route
G1
G2
Medium importance of gift
High importance of predefined route
G1
G2
Medium importance of gift
Low importance of predefined route

Figure 4.16: *Generated recommendation policies as a function of $\mathcal{U}$'s ability to follow recommendations correctly and $\mathcal{U}$'s desire to find a present. The importance to follow a predefined route is taken into account.*
(Dark and light arrows represent recommendations given in speech mode and map mode, respectively. Dashed lines show where movements are possible but not recommended. The width of an arrow or a line between two locations reflects the number of shops between them.)

fixed deadline is an extraneous restriction that cannot be bypassed by the user. That is, there is no alternative to leaving the shopping mall at the fixed deadline.

In the airport scenario, we can assume a soft deadline for the navigation task only if it is guaranteed that the passenger makes it to the gate in time anyway, that is, if arriving late at the gate can at most, for example, cause a bad seat assignment but not missing the flight. This assumption requires that the system can determine in advance if a deadline can never be exceeded, no matter what happens. However, for uncertain processes, this guarantee can never be given. At best, a system can guarantee to make a deadline with a particular probability.

In what scenarios are deadlines truly relevant? The airport scenario is possibly the most striking example, as the result of not meeting the deadline—missing the flight—represents an unpleasant experience that one would like to avoid at any price. This is exactly what matters where a fixed deadline is involved: the price of not meeting the deadline. Missing a flight has such a high cost (e.g., the cost of not getting to a conference because no backup-flight is available—or only at a tremendous price) that there is usually nothing to do in an airport worth taking the risk of missing the flight (i.e., there is usually nothing important enough on the way to the gate that could be traded-off against missing a flight). This assumption implies that whether a deadline is fixed or not depends on the proportion of costs and rewards.

We will discuss two possible approaches to solve problems that involve a task to be finished before a fixed deadline. The first one, *finite-horizon decision-theoretic planning*, is rather easy to implement, the second one, *time-dependent Markov decision processes* (Boyan & Littman, 2000), is more general than the first one. We will illustrate the first approach in the context of a system giving navigation recommendations in an airport building, thereby taking the fixed deadline to get to the gate into account.

4.4.1 Finite-Horizon Decision-Theoretic Planning

The key idea of the approach that we have chosen to deal with fixed deadlines is to keep different policies ready for different available portions of time until a given deadline. Thereby, we face the following problem: In the basic MDP framework, a process evolves in stages that do not correspond to any particular amount of time. The execution of each action corresponds to the proceeding of the process by one stage—no matter how much time elapses during the execution of the action. If all actions considered in an MDP took the same amount of time (say one time unit), n stages of a process (or the execution of n arbitrary actions from the set of actions $\mathcal{A}$) would correspond to n time units elapsing. As there is no such restriction on the time that elapses during the execution of a particular action, the duration of n stages is in general not predictable in the basic MDP framework. To solve the fixed deadline planning problem with finite-horizon planning, we have to establish a relation between stages and the duration of actions.

In the original MDP framework, each n-stage-to-go policy (cf. Section 3.1.4) represents the optimal policy for the situation in which the interaction between the system and the user does not exceed n actions of arbitrary length. Each execution of an action corresponds to the decision process proceeding exactly one stage.

Instead of allowing stages to be arbitrarily long, we now define each stage to correspond to exactly one time unit. We chose the duration of that time unit in a way that the duration of each action $a \in \mathcal{A}$ can be expressed as a multiple of that time unit. Now, we can define the t-time-units-to-go value function as $V_0(s) = 0$ and

$$V_t(s) = \max_{a \in \mathcal{A}} \{ \texttt{if } d(a) \leq t \texttt{ then } C(a) + \sum_{s' \in \mathcal{S}} P(s' \mid a, s) V_{t-d(a)}(s') \texttt{ else } 0 \} \tag{4.2}$$

with $d(a)$ the duration[8] of action a. As a consequence of this definition, actions that take longer than t time units ($d(a) > t$) will not be considered at stage t at all. The policy

$$\pi_t = \arg\max_{a \in \mathcal{A}} \{ \texttt{if } d(a) \leq t \texttt{ then } C(a) + \sum_{s' \in \mathcal{S}} P(s' \mid a, s) V_{t-d(a)}(s') \texttt{ else } nil \} \tag{4.3}$$

represents the optimal t-time-units-to-go-policy (with nil corresponding to "no action available that allows to reach the goal before the deadline"). We will illustrate the approach with a simplified version of the navigation assistance problem described in Section 4.2.2.

Example We assume that the airport assistance system $\mathcal{S}$ wants to show passenger $\mathcal{U}$ the way to the gate and consider a single secondary tasks, for example, buying a gift along the way. Suppose that $\mathcal{U}$ is at Location B and wants to get to gate G1.

In this context, it is convenient to use positive integers to denote points in time such that lower integers represent later points in time and 0 denotes the deadline (i.e., in the scenario at hand the last point in time to get on board of the plane). Figure 4.17 shows the t-step-to-go policies for $t = 0, 1, 3, 5$, and 15. If $\mathcal{U}$ is still at Location B for $t = 0$, then he is just about to miss his flight. For $t < 5$, $\mathcal{S}$ has no recommendation available for $\mathcal{U}$ at Location B because it is not possible for $\mathcal{S}$ to assist $\mathcal{U}$ in a way that $\mathcal{U}$ gets from Location B to the gate in time. These are situation that will not occur if (1) the interaction between $\mathcal{S}$ and $\mathcal{U}$ starts early enough such that $\mathcal{U}$ can at least in principle reach his departure gate in time and (2) $\mathcal{S}$'s model is accurate in that, for example, it describes correctly how much time $\mathcal{U}$ needs to get from one place to another. For this simple example, we assume that $\mathcal{U}$ can reliably follow $\mathcal{S}$'s recommendations, that is, he never ends up in a wrong place when following a recommendation.

For $t \geq 5$, $\mathcal{S}$ is able to give $\mathcal{U}$ a recommendation that allows $\mathcal{U}$ to reach the gate from Location B before the end of the boarding. $\mathcal{S}$ sends $\mathcal{U}$ to gate G1 via Location E, although this is not the shortest path—we assume that the length of the arrows is proportional to the distances between two places. The reason for the duration that $\mathcal{U}$ needs from Location E to Location F being much shorter than the duration that he needs to go from Location B to Location D (although the distance being equally long) might be, for example, that $\mathcal{U}$ can step onto some sort of conveyor belt, as they are common in large airports.

The approach also takes into account different durations for $\mathcal{U}$ to get from one place to the next dependent on the recommendation mode that $\mathcal{S}$ has chosen. For example, one can specify that to get from one location to another, $\mathcal{U}$ will need one additional time unit for each shop along the way that he is made aware of by being recommended in map mode instead of speech mode. Additionally, the time that it takes $\mathcal{U}$ to buy something in a shop has to be considered in the model.

For $t = 15$, $\mathcal{S}$ sends $\mathcal{U}$ towards the gate G1 via Location C and makes him aware of the shops along the way by selecting map mode to present the navigation recommendations. There is enough time to select the longer way (in terms of duration, not distance) for $\mathcal{U}$ and make him aware of shops, thereby inviting him to spend time on shopping.

[8]The duration in this approach is assumed to be constant and certain, while the approach by Boyan and Littman (2000) allows varying, uncertain action durations.

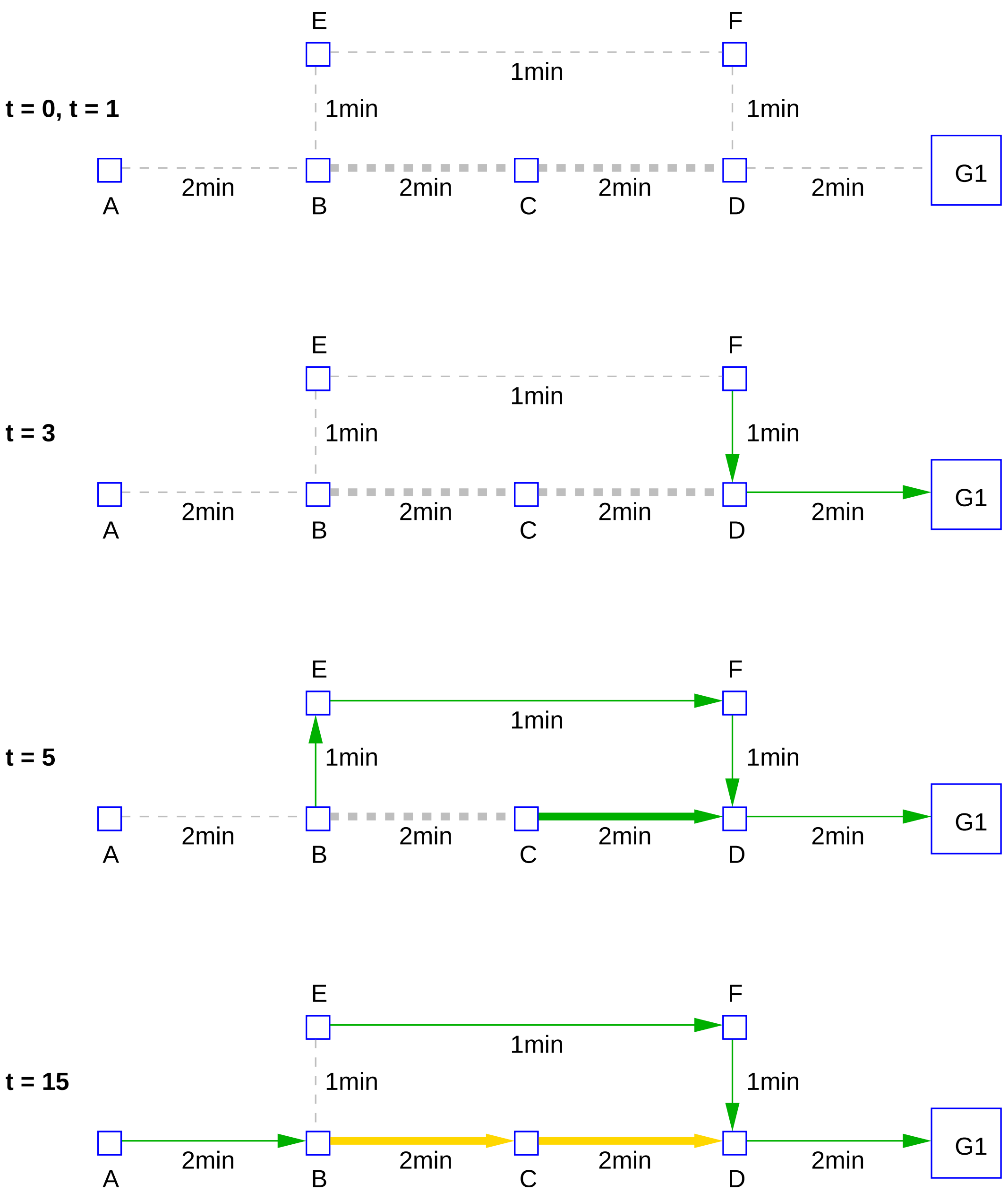


Figure 4.17: *Finite-horizon policies for a simple navigation problem.*
(The graphs shows parts of selected t-time-units-to-go policies (with 1 time unit = 1min). Green arrows mean that speech mode is used, yellow arrows mean that map mode is used. The thickness of the arrows/lines corresponds to the probability of fi nding a gift on the corresponding part of the route. Although longer in distance, the path from Location B *to gate* G1 *via* Location E *is faster than the path via* Location C*. Therefore, the system tends to send the user via* Location E *to the gate if the deadline (boarding) is not far in the future anymore.)*

We have implemented the finite-horizon decision-theoretic planning approach for the scenario of high complexity in Chapter 5 (see in particular Section 5.3). The space required to store the finite-horizon policies grows linearly in the number of time steps if coded in a naive way. The representation of the finite-horizon policies can be compressed by deriving time step intervals for which particular recommendations are given.

4.4.2 Boyan & Littman (2000): Time-Dependent MDP Approach

Boyan and Littman (2000) introduce the *time-dependent MDP (TMDP)* model, which includes both stochastic state transitions and stochastic, time-dependent action durations (action costs). In contrast to the approach described in the last section, this allows to consider problems in which the costs for transitions or the rewards for goal states vary over time. A TMDP is a special continuous-state Markov decision process (see, e.g., Puterman, 1994) consisting of states with both a discrete component and a real-valued time component.

The TMDP model is motivated by a scenario that involves decisions about optional means of transportation. Buses, trains, and subways vary in their expected travel time according to the time of day: Buses and subways come more frequently during the rush hour; trains leave on or close to scheduled departure times; highway driving times vary with time of day, with heavier traffic and longer travel times during the rush hour.

The TMDP model enhances the MDP model in the following way: In addition to a discrete state space $\mathcal{S}$ and a discrete action space $\mathcal{A}$, Boyan and Littman (2000) introduce a discrete set of outcomes $\mathcal{M}$, each of the form $\mu =< x'_\mu, T_\mu, P_\mu >$ with

- $x'_\mu \in \mathcal{S}$: the resulting state,
- $T_\mu \in \{\text{ABS}, \text{REL}\}$: specifies the type of the resulting time distribution, and
- $P_\mu(t')$ if $T_\mu = \text{ABS}$: probability distribution over absolute arrival times of μ, or
- $P_\mu(\delta')$ if $T_\mu = \text{REL}$: probability distribution over durations of μ.

Moreover, they define

- $L(\mu \mid x, t, a)$ as the likelihood of outcome μ given state x, time t, and action a,
- $R(\mu, t, \delta)$ as the reward for outcome μ at time t with duration δ, and
- $K(x, t)$ the reward rate for “dawdling” in state x at time t.

The optimal value function for a TMDP in terms of these quantities is then defined with the following *Bellman equations*:

$$V(x,t) = \sup_{t' \geq t} \left(\int_t^{t'} K(x,s)ds + \overline{V}(x,t') \right) \tag{4.4}$$

$$\overline{V}(x,t) = \max_{a \in \mathcal{A}} Q(x,t,a), \tag{4.5}$$

where

$$Q(x,t,a) = \sum_{\mu \in \mathcal{M}} L(\mu \mid x,a,t) U(\mu,t) \tag{4.6}$$

$$U(\mu,t) = \begin{cases} \int_{-\infty}^{\infty} P_\mu(t')\,[R(\mu,t,t'-t) + V(x'_\mu,t')]dt' & \text{if } T_\mu = \text{ABS} \\ \int_{-\infty}^{\infty} P_\mu(t'-t)\,[R(\mu,t,t'-t) + V(x'_\mu,t')]dt' & \text{if } T_\mu = \text{REL}. \end{cases} \tag{4.7}$$

The solution to a TMDP is a policy mapping state-time pairs $< x, t >$ to actions. The time-value functions for each state can be arbitrarily complex in the general TMDP model. It is impossible to represent them exactly, but the model can be restricted in that the time-value function $V_i(t)$ is represented as a piecewise linear function of time, which allows value functions to be manipulated exactly.

Rewards must be constrained to be piecewise linear functions of start and arrival times and action durations: $R(\mu,t,\delta) = R_s(\mu,t) + R_a(\mu,t+\delta) + R_d(\mu,\delta)$ where R_s, R_a, and R_d are piecewise linear functions of start time, arrival time, and duration, respectively. In addition, the dawdling reward K and the outcome probability function L must be piecewise constant. Arrival and duration probability distribution functions must be discrete.

In the context of the navigation assistance example, representing absolute time as part of the state space allows to consider the value of a navigation recommendations dependent on the time of day. For example, the value of the recommendation to pass a narrow corridor might be smaller at times when the corresponding part of the airport building is congested in the morning hours, or the recommendation to go to a particular shop might be more valuable if it is known that the time to stand in line at the cash desk is short in the afternoon. Similarly, the recommendation to "dawdle" in the current area because the shops will open in some minutes time can receive higher value than going somewhere else if the decision-theoretic model can take the shops' opening hours into account. What looks attractive at first sight also poses a problem: A lot more and more complex data has to be provided to initialize a TMDP model. Moreover, the implementation of a TMDP model involves the application of the Bellman Equations 4.4, 4.5, 4.6, and 4.7, which are more complex than the standard Bellman equations (cf. Equation 3.2). We have not implemented the TMDP approach for the navigation assistance example but assumed constant action durations instead.

Some objectives that can be achieved by considering absolute time as part of the state space can be approximated well by the finite-horizon approach or are not desirable in the scenarios that we consider. For example, minimizing the expected time for getting to the gate or maximizing the probability of making it to the gate in time might not be convenient if it involves unnecessarily wasting time at the gate that could be spent more conveniently (e.g., by some shopping or having a snack in a fast food restaurant).

4.5 Synopsis

In this chapter, we have explained the decision-theoretic models and their application in two scenarios with comparatively low complexity. The first scenario—providing assistance for operating a technical device—is a typical example for how decision-theoretic planning can contribute to a system's adaptation capabilities. We have described how variations of the form of output presentation along different dimensions (in particular the bundling size of instructions, but also decisions about speech versus text output or short versus verbose instructions) can be based on decision-theoretic planning and thereby on the expected utility for a strategy. The interplay of the decision-theoretic planner and the user modeling component, as we have described it in the context of the READY prototype, allows the system to adapt to the user's resource limitations (i.e., the user's subjectively felt time pressure and his cognitive load).

The second part of the chapter focused on adaptive navigation assistance in complex buildings. In this context, adaptivity with regard to the system's output presentation cannot adequately be achieved without taking the spatio-temporal constraints of the complete navigation task into account. We have shown how navigation planning and presentation planning can elegantly be combined in a single decision-theoretic planning process, which can again be parameterized by a user modeling component providing estimations of the user's time pressure and cognitive load. The integrated approach accounts for the navigation task and the presentation task influencing each other. However, if—for example, for modularity reasons—such integration is deprecated, the decision-theoretic model that we have described can without much effort be modified (and sometimes simplified) such that the decision-theoretic planner exclusively solves the navigation task or the presentation task. The latter has been explained in the context of combining predefined routes with decision-theoretic planning. Eventually, we have used the navigation task to describe how fixed deadlines can be dealt with in a decision-theoretic planning approach. The approach that we have chosen to deal with fixed deadlines turns out to be helpful to manage more complex scenarios, as they will be the topic of the next chapter.

5 FOMDPs in Scenarios of High Complexity

The two examples that we have used in the previous chapter to illustrate the development of a decision-theoretic model for an intelligent adaptive user interface planning problem are both low in complexity. To restrict the complexity of the decision-theoretic planning approach, we have committed to some simplifications in the problem descriptions. For example, for the navigation assistance problem (see Section 4.2.2), we have assumed—apart from a simplified airport terminal topology—that the user has exactly one navigation goal (getting to the departure gate) and one subordinated goal (buying a gift) that he would like to achieve on the way to the gate.

In this chapter, we explore approaches to deal with a class of more realistic and hence more complex instances of the navigation assistance problem. The complex example problem is based on a more realistic airport terminal and involves several navigation goals as well as several subordinated goals that the user would like to achieve in parallel with the navigation tasks. Later in this chapter, we will discuss the classification of goals not only as primary and secondary, but more generally as *n-ary goals*.

Much work has focused on factored representations (see Section 3.1.5.1) to deal with large Markov decision problems. However, approaches based on factored representations, such as those introduced, for example, by Hoey et al. (1999) or Feng and Hansen (2002), do in general not scale well enough to deal with realistic problems[1].

Plutowski (2003) reports an evaluation of four MDP solvers based on factored representations (cf. Section 3.1.5.1) in a shopping scenario similar to the one described in Chapter 7. Moreover, he introduces a composite MDP solver that draws from each of the four techniques. With the composite MDP solver, it is feasible to solve moderately sized task instances from the class of problems described in Chapter 7. In addition to the modeling that we employed for the user studies of Section 7.2 and 7.3, the decision-theoretic model used by Plutowski considers the information which stores have already been visited. This additional information blows up the decision-theoretic model in the same way as the information about which items have already been bought. Therefore, an MDP encoded as described by Plutowski for a problem with 13 locations and 7 items to be bought requires already 17 billion states.

The second approach to deal with complexity in decision-theoretic planning is hierarchical decomposition: Complex decision-theoretic planning problems are decomposed into less com-

[1]For example, we have mentioned in Section 3.1.5.1 that SPUDD has been demonstrated on a class of problems involving up to 63 million states; this may sound a lot but in fact is not, considering that the state space grows exponentially in the number of binary state features.

plex decision-theoretic planning problems. We will survey some examples of decomposition approaches in the next section. Decomposition has advantages and disadvantages: It allows to deal with more complex problems at the price of requiring additional domain knowledge and typically only approximating the optimal solution of a decision-theoretic planning problem.

The decomposition approaches that we will survey in the next section are not suitable for the kind of problems that we are interested in. None of them exploits hard time constraints and the potential of distinguishing different types of goals. We will introduce a new approach of hierarchical decomposition which takes advantage of the existence of hard time constraints (as they were described in Section 4.4) and different types of goals that occur in domains like the airport domain introduced in Section 4.2.2. The hierarchical approach can be characterized as *goal-priorized decision-theoretic planning (GPDTP)* with Markov decision processes on n levels. Thereby, the decision-theoretic planning processes on levels n and $n+1$ can pairwise by considered as *macro-level* and *micro-level decision-theoretic planning processes*. The approach—apart from enabling us to deal with complex realistic problems—has positive side effects for the system's usability and its capability to consider partial observable information appropriately without explicitely modeling the interaction as POMDP. The example that we employ throughout this chapter distinguishes only primary and secondary goals, hence requires two-level goal-priorized decision-theoretic planning, only.

5.1 Hierarchical Decomposition

Hierarchical decomposition (Sacerdoti., 1974) is a well known concept in the classical planning paradigm (for a survey, see, e.g., Ghallab et al., 2004, Part III, Chapter 11, or Yang, 1997). The key benefit of hierarchical decomposition in general is that, at each hierarchy, a computational task is reduced to a (small) number of simpler subtasks at the next lower level to downsize the computational cost for finding a solution to the top-level problem. In the best case, hierarchical methods can result in linear-time instead of exponential-time planning algorithms (see, e.g., the comments by Russell & Norvig, 2003, in their survey of hierarchical planning approaches).

Much of the effort in the context of decomposition has focused on methods that decompose MDPs into "communicating" subproblems (Bertsekas & Tsitsiklis, 1989). Typically, a Markov decision process is either already specified in terms of a set of "pseudo-independent" subprocesses (see, e.g., Singh & Cohn, 1998) or is automatically decomposed into such subprocesses (see, e.g., Boutilier, Brafman, & Geib, 1997).

Meuleau et al. (1998) describe an approach for solving very large *weakly coupled Markov decision processes* in which they exploit two key properties of particular planning problems: (1) The problems are composed of multiple tasks whose utilities are independent, and (2) the actions taken with respect to a task do not influence the status of any other task. They model these tasks as Markov decision processes which are weakly coupled by resource constraints in that actions selected for one MDP restrict the actions available to other MDPs. The solutions of the sub-MDPs are merged or used to construct an approximate global solution.

The method of Meuleau et al. (1998) is based on so called *Markov task sets*, which are approximated by a strategy they call *Markov task decomposition*. Markov task decomposition is divided into two phases: (1) In an offline phase, value functions are calculated for the individual tasks using dynamic programming. (2) In an online phase, these value functions are used to calculate the next action as a function of the current state of all processes. Meuleau et al. (1998) apply their approach with encouraging results in domains such as job shop scheduling, allocation of repair crews to different jobs, and disaster relief scheduling. Similar approaches for weakly coupled Markov decision processes are explained by Dean and Lin (1995), who in particular introduce two algorithms for the generation of a global solution from the solutions of the subproblems, and Parr (1998), who emphasizes the benefit of decomposing weakly coupled Markov decision processes for problems with similar substructures.

Hauskrecht, Meuleau, Kaelbling, Dean, and Boutilier (1998) investigate the use of *macro-actions*[2] to build abstract MDPs that approximate the original MDP and can be solved more efficiently. They propose a hierarchical model that works with macro-actions only and thereby significantly reduces the size of the state space. Their key idea is to treat macro-actions as local policies that act in certain regions of the state space and thereby restrict the states in the abstract MDP to those at the boundaries of regions in the original MDP. This way, they construct models on the basis of macro-actions that allow the macro-actions to be treated as if they were ordinary actions in the original MDP. That is, a macro-action for a region is a local policy for this region. A similar approach is described by Lane and Kaelbling (2001), where each macro represents knowledge about how to achieve one particular sub-goal within nearly-independent regions of the state space. Hauskrecht et al. (1998) apply their approach in a grid-world maze domain, where they achieve a remarkable reduction in the size of state and action spaces. They also emphasize the advantage of potential reuse of macros to solve multiple related MDPs.

While most hierarchical MDP algorithms rely on hand-designed, knowledge-intensive state abstraction, Pineau et al. (2003) propose an algorithm which plans from the bottom of the hierarchy up and thereby automatically finds good abstractions contingent on their subtasks' optimal policies. They claim that this so called policy-contingent approach can often abstract away more details than a human can, because it has access to its subtasks' policy when it decides whether to cluster states. The approach is also applicable to partially observable MDPs. Pineau et al. (2003) test their approach in the taxi domain (Dietterich, 2000), a commonly used example problem in the hierarchical MDP literature, where the overall task is to control a taxi agent with the goal of picking up a passenger from an initial location and then dropping him off at a desired destination. In this domain, their approach outperforms comparable approaches such as MAXQ (Dietterich, 2000) with regard to the parameters required to learn the solution.

Also Moore, Baird, and Kaelbling (1999) try to automate the decomposition process. They introduce an approach for automatically generating action hierarchies for Markov decision processes with multiple goals. They present an algorithm that defines landmarks in the state space and thereby partitions the state space automatically. They tested their algorithm in grid-world mazes of different size.

Region-based decomposition has also been studied in the context of plan recognition. Bui, Venkatesh, and West (2000) describe how *abstract Markov policies* can be recognized from external observations, for example, to track and predict human movement in large spatial environments.

5.2 Example Domain

In the example of Section 4.2.2, we assume that the passenger needs navigation assistance to the gate and wants to buy a gift along the way. This assumption simplifies the procedure that we

[2]For deterministic action models, *macro operators* have been studied, e.g., by Korf (1985).

usually have to go through before we are seated in the correct air plane, once we have entered the airport building.

Typically, one of the first things that we want to do after arrival at the airport is to get rid of our luggage, that is, to go to one of the assigned check-in counters rather quickly. Already on the way to the check-in counter, we might bear in mind desires such as finding a rest room, completing our travel pharmacy with things that we did not have the time to buy before our trip, having a snack in a fast food restaurant and maybe a coffee in a café, buying duty-free items or some other gift item for the friend who we are visiting, buying some batteries for our new digital camera, a book or a magazine to read the flight, or some mint drops to chew during the landing. Usually, one would not try to achieve all subordinated goals right in the beginning. But if the situation is adequate, some of the goals on the to-do-list can already be achieved on the way to the check-in counter, for example, if the passenger happens to come along some kiosk or rest room. Other desires might typically be left to be considered after the passenger has already checked in and does not have to carry his luggage anymore, for example, buying a book, the batteries for the camera, or some item for the travel pharmacy. And eventually, some items would be considered after the passenger has already passed the passport control, for example, the mint drops or items to be bought in a duty-free shop.

The list of subordinated goals that a passenger wants to achieve while he goes through the standard procedure at the airport (checking in, going through the passport control, and going to the gate) might in general not be huge but sometimes can get to a considerable size, in particular, if several people, for example a whole family, are traveling together. To consider an increasing list of navigation goals and desires that a passenger might want to be considered by the system during his stay in the airport terminal in a single decision-theoretic model has two major disadvantages: (1) The size of such a decision-theoretic model is hardly manageable and (2) the system behavior resulting from such a decision-theoretic model is rather intransparent, which means that the usability of a system employing such a complex decision-theoretic model will suffer. Usability issues will be the topic of Chapter 7.

5.2.1 Goal Priorization: Primary and Secondary Goals

As has already been indicated, our decomposition approach for the airport domain at hand is based on the distinction of two classes of goals to be achieved by the decision-theoretic planning approach: (1) obligatory or *primary goals* and (2) optional or *secondary goals*. Of both classes, we have already introduced one particular instance in the example problem of Section 4.2.2. There, the primary goal is getting to the gate, and the secondary goal is buying a gift on the way to the gate if possible. We have already indicated what other primary and secondary goals typically exist in the airport domain. As primary goals, we have identified the physical locations in the airport terminal, which the passenger is obliged to go to at some point: the check-in counter, the passport control, and eventually the gate.

In the airport domain, the order in which the primary goals have to be achieved is fixed. In other domains it can be flexible. Consider, for example, a patient's trip through a hospital to get through preliminary check-ups for a medical treatment before lunch time[3]. Here, the order in which to visit, for example, the lab for the blood test, the x-ray cabin, and the audiometry lab, might be flexible. In this example, getting to the corresponding labs in the hospital can be

[3]We say "before lunch time", because the time restriction has consequences for our decomposition approach; secondary goals in this scenario might be buying a magazine or a phone card while going from lab to lab.

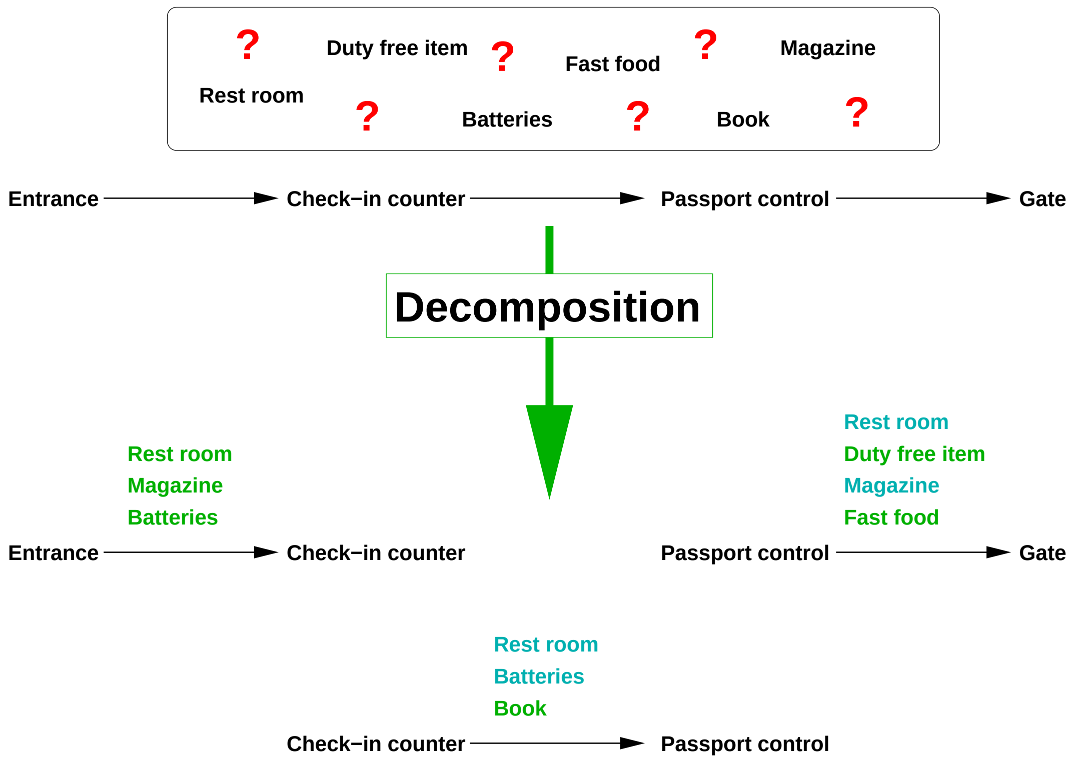


***Figure 5.1**: Decomposition of a complex decision-theoretic planning problem.*
(The considered instance of a complex decision-theoretic planning problem involves three navigation goals and several subordinated goals to be achieved during the navigation tasks. The decomposed decision-theoretic planning problems involve one navigation goal each and only a subset of possible subordinated goals to be achieved during the navigation task. Blue labels: Subordinated goals that could not be achieved can be considered in later micro-level decision-theoretic planning processes again.)

considered as primary goals that can be achieved in arbitrary order. Flexibility with regard to the primary goal order has no consequence for the applicability of our decomposition approach in principle. However, it causes an increased complexity of the macro-level decision-theoretic planning process, as it requires additional state features in the macro-level MDP, including the information which primary goals have already been achieved.

Apart from buying a gift, we have already mentioned other potential secondary goals in the airport domain, for example, visiting a rest room, having a snack in a fast food restaurant, or buying various articles before the departure. They all have in common that the passenger usually tries to achieve them only if there is enough time to do so. Ideally, they passenger would like to achieve all secondary goals, but in general, he can only try to achieve as many of them as possible in the time that remains before the boarding. The order in which secondary goals are achieved is usually arbitrary, but the passenger might consider some secondary goals more important than others. For example, finding a rest room might be assigned hight importance in some situations. However, strictly trying to achieve secondary goals in the order of their importance might not be appropriate. If an important secondary goal can with high probability be achieved rather shortly before the boarding (e.g., buying a duty-free item), while a less important secondary goal can be achieved early in the process (e.g., buying a magazine for the flight), then trying to achieve the less important secondary goal before the more important one will typically be considered to be appropriate. Even if not all secondary goals can be achieved in the given time, the passenger might not necessarily want to consider the most important secondary goal first. For example, if the most important secondary goal is difficult (unlikely) to be achieved, the passenger might prefer to try to achieve two less important secondary goals whose achievement is more likely.

5.2.2 Decomposition and Time Restrictions

The airport domain involves the consideration of a fixed deadline: The passenger must reach the gate in time. What does this fixed deadline mean for time restrictions of preceding primary goals—getting to the check-in counter and going through the passport control? How do the time restrictions for the complete process (from entering the airport building to boarding) influence the time restrictions for each particular primary goal?

If we can determine appropriate time restrictions for the achievement of each primary goal, we can consider the achievement of each of the primary goals as a separate finite-horizon decision-theoretic planning problem that can be solved as described in Section 4.4. Considering several secondary goals while trying to achieve a particular primary goal turns out not to be difficult, as we will explain in Section 5.3.

However, it is not trivial to derive the time restrictions for each particular primary goal from the fixed deadline for reaching the gate. A naive approach, such as assigning the same amount of time to each primary goal, does not seem to be appropriate. The distances to get from one location to another (e.g., from the entry to the check-in and from the check-in to the passport control) can be very different, and so can the number of secondary goals that can potentially be achieved (and hence the time spent trying to achieve these secondary goals) on different sections of the way to the gate. The hierarchical decision-theoretic planning approach described in the next section incorporates time restrictions in an elegant way into the decomposition process and considers them adequately by interleaving the macro-level policy and the micro-level policies.

5.3 Goal-Priorized Decision-Theoretic Planning

In this section, we introduce goal-priorized decision-theoretic planning (GPDTP), a new form of hierarchical decision-theoretic planning with fixed deadlines. The airport domain requires only the distinction of primary and secondary goals—the general case with a more complex goal priorization will be discussed in Chapter 6.

We illustrate the macro- and micro-level decision-theoretic models for the airport domain and explain how the solution to the global problem evolves from the solutions generated on the micro-level. The two key insights involve the benefit of taking advantage of the existence of a fixed deadline for the overall planning goal and distinguishing two types of goals that are present in the airport domain. The idea is to assign an appropriate subset of secondary goals and appropriate portions of time to the achievement of each of the primary goals. Thereby, the performance of the user in one section (trying to achieve one primary goal) can be taken into account with regard to the system's recommendations in succeeding sections (trying to achieve other primary goals). Using decision-theoretic planning on a macro- and a micro-level of the planning process allows to take these dependencies into account without requiring any replanning.

5.3.1 Preliminary Computations

On the micro-level, the airport domain is specified in terms of the distinct physical locations in the airport building, that is, the positions at which infrared beacons would be installed in the physical environment to enable the system to localize the user. Each physical location has an identification number, and we know for each physical location what its adjacent locations are. For each pair of adjacent locations, we know the physical distance between the two locations—from which we can derive the time needed to get from one location to another for any individual user—and the probability that the user achieves any of the secondary goals on the way from one location to another. Moreover, each physical location is assigned to a distinct area of the airport building. In our example implementation, we have studied an airport building with four areas: two public areas and two non-public areas. A user is in a public area before he passes the passport control. A public area of a terminal building usually hosts check-in counters, stores, cafés, fast food restaurants, rest rooms and the like. A passenger is in a non-public area of a terminal building after he has passed the passport control. Non-public areas host, for example, duty-free shops, cafes, rest rooms and eventually the gates. In our example implementation, each physical location in the airport building is assigned to one out of the four distinct areas.

The preliminary computations for the macro-level decision-theoretic planning includes three steps: (1) the determination of the regions that are involved to achieve the next primary (or navigation) goal, (2) the partitioning of the total time, that is, a rough estimation of appropriate portions of the available total time for the achievement of each of the primary goals, and (3) the derivation of macro-level probabilities from micro-level probabilities, that is, an assessment of the probability for each secondary goal to be achieved by the user in any particular section of the way to the gate (i.e., together with any of the primary goals).

Determination of the regions: The determination of the regions that are involved to get to the next navigation goal (i.e., to achieve the next primary goal) is not difficult. A straightforward approach is to determine the shortest path from one location to another and check for all locations on the path to which areas they belong to. The micro-level decision-theoretic planning process must cover all areas identified for the primary goal at hand. Alternatively, for example, the most likely or the most common path can be considered. In the example implementation, we decided to consider the shortest path, which can be computed with *Dijkstra's algorithm* (Dijkstra, 1959, or see, e.g., Sedgewick, 2003, Chapter 21).

Partitioning the total time: We have applied the following heuristics for the estimation of appropriate portions of the available total time for the achievement of each of the primary goals: First, we compute for each primary goal the minimal time that is required for its achievement. In our example, this includes the minimal time to get from the current position to the check-in counter, the minimal time to get from the check-in counter to the passport control, and eventually the minimal time to get from the passport control to the gate. Then, we partition the total time proportionally to the minimally required times for achieving each of the primary goals to obtain reasonable maximal times to achieve each of the primary goals. Note that these times are only used in the macro-level decision-theoretic planning process. Alternative heuristics might be considered under particular circumstances, for example, a heuristic taking the number of opportunities to achieve particular secondary goals in a particular section of the airport building into account. It might also be possible to learn from empirical data acquired about the proportions of time spent by a number of passengers to achieve the typical primary goals before departure.

The micro-level decision-theoretic planning processes have to take into account that more time than the originally estimated maximal time for the achievement of a primary goal can be available because preceding primary goals have been achieved faster than preliminarily estimated. We say more about how time is dealt with in the micro-level decision-theoretic planning processes in Section 5.3.3.

Derivation of macro-level costs and probabilities from micro-level costs and probabilities: The costs of macro-level actions are composed of the time needed for covering the distance to achieve a particular primary goal and the duration(s) to achieve the secondary goal(s) considered by the macro-level action at hand (see also Figure 5.3).

The last piece of information that we need for the decision-theoretic planning on the macro-level is the probability that a particular secondary goal is achieved together with any of the primary goals. The idea is to estimate this probability from the probabilities that are used on the micro-level, that is, the probabilities to achieve a particular secondary goal on the way from one distinct location[4] to an adjacent distinct location. We determine these probabilities heuristically: An approximation for the probability $p_G(g)$ to achieve the secondary goal g together with the primary goal G can be computed from the total number of possibilities $n_r(g)$ to achieve the secondary goal g in any of the regions r from the set of regions R_G that are relevant for the achievement of the primary goal G and the total length $l(r)$ of all paths that make up the regions $r \in R_G$:

$$p_G(g) = \texttt{if } \frac{\sum_{r \in R_G} l(r)}{10 \sum_{r \in R_G} n_r(g)} < 1 \texttt{ then } \frac{\sum_{r \in R_G} l(r)}{10 \sum_{r \in R_G} n_r(g)} \texttt{ else } 0.99 \tag{5.1}$$

This heuristic can be interpreted as follows: The more possibilities to achieve a particular secondary goal exist in the region(s) corresponding to the considered primary goal and the closer the corresponding locations lie together, the higher is the probability that the secondary goal is actually achieved together with the primary goal at hand. This heuristic yields an intuitive system behavior in the example implementation. The macro policy suggests to consider secondary goals with a reasonable chance for being achieved if the remaining time allows. However, other heuristics might do similarly well.

5.3.2 Decision-Theoretic Planning on the Macro Level

The preliminary computations provide all quantitative information that we need to build an appropriate decision-theoretic model for the macro-level planning and to compute a *macro-level policy*. A macro-level policy in this context can be considered as a time-dependent assignment of sets of secondary goals to primary goals. It consists of directions of the following form: If while starting to work on primary goal p_1 at time t only s_2 of the set of all secondary goals $\{s_1, s_2, s_3, s_4\}$ has been achieved, then try to achieve s_1 and s_3 together with the primary goal at hand (i.e., consider s_1 and s_3 in the micro-level decision-theoretic planning process for the primary goal at hand). With equally important secondary goals and decreasing remaining time to achieve the primary

[4]Recall that the distinct locations on the micro-level are those that correspond to a position of an infrared beacon. Distinct locations on the macro-level are locations such as the initial position of the user, the check-in counter, the passport control, or the gate.

goal p_2, the set of secondary goals to be considered together with p_2 will typically become smaller and secondary goals consuming much time will be replaced by secondary goals consuming less time. In particular, if the time assigned to the achievement of a primary goal is running out, then the macro-level policy will eventually assign no secondary goal at all to the primary goal at hand. Figure 5.2 sketches a part of the macro-level t-time-units-to-go policies for $t = 25, 15$, and 5.

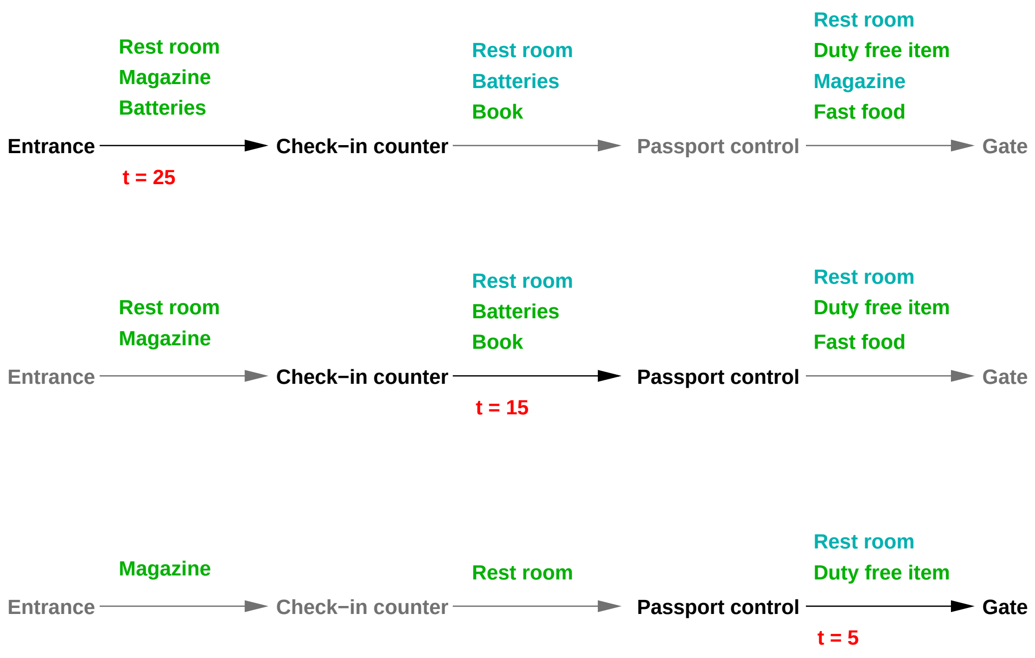


***Figure 5.2**: Decomposition approach dealing with fixed deadlines—sketch of a macro policy.*
(The macro policy specifies dependent on the remaining time—we consider the finite-horizon policies with 25, 15, and 5 minutes until the gate is closed—which subset of all secondary goals is supposed to be considered in the micro decision-theoretic planning processes in different sections of the passenger's way to the gate. For example, in the first section—entrance to check-in counter—with 25 minutes remaining, the system will only take into account assistance for finding a rest room, a magazine, and batteries. If not being achieved in one section, some secondary goals can be reconsidered in other sections, presuming that there is enough time to do so. The closer it gets towards the fixed deadline, the less secondary goals will the macro policy assign to be considered in the micro decision-theoretic planning processes.)

In the decision-theoretic model underlying the macro-level planning process, the states are described by the important macro-level locations (the entrance of the airport building[5], check-in counter, passport control, and gate[6]) and the information about which of the secondary goals have already been achieved. The latter piece of information is not included in Figure 5.2.

Figure 5.3 shows a fraction of the macro-level decision-theoretic model: the transitions from the state describing the situation that the user's next primary goal is the gate and two out of four secondary goals have already been achieved. We only regard the action *consider duty-free shops*

[5]For the preliminary computation of the quantitative information used for the macro-level planning process, the system needs to know the initial position of the user; for the given example problem, we assume that the initial position is the entrance of the airport building; however, any other location as initial position would work equally well.

[6]Check-in counter, passport control, and gate have to be specified exactly in the concrete example, e.g., check-in counter 121, passport control in hall B, and gate B37

and rest rooms as secondary goals on the way from the passport control to the gate. The action yields four transitions which are associated with costs and probabilities that are computed as described in Section 5.3.1. In the figure, we assume the situation in which the passenger has already achieved two secondary goals (buying batteries and having a snack in a fast food restaurant) when leaving the passport control. The figure then shows the transitions for the action *consider duty-free shops and rest rooms as secondary goals* on the way from the passport control to the gate, that is, while trying to achieve the last one of the primary goals.

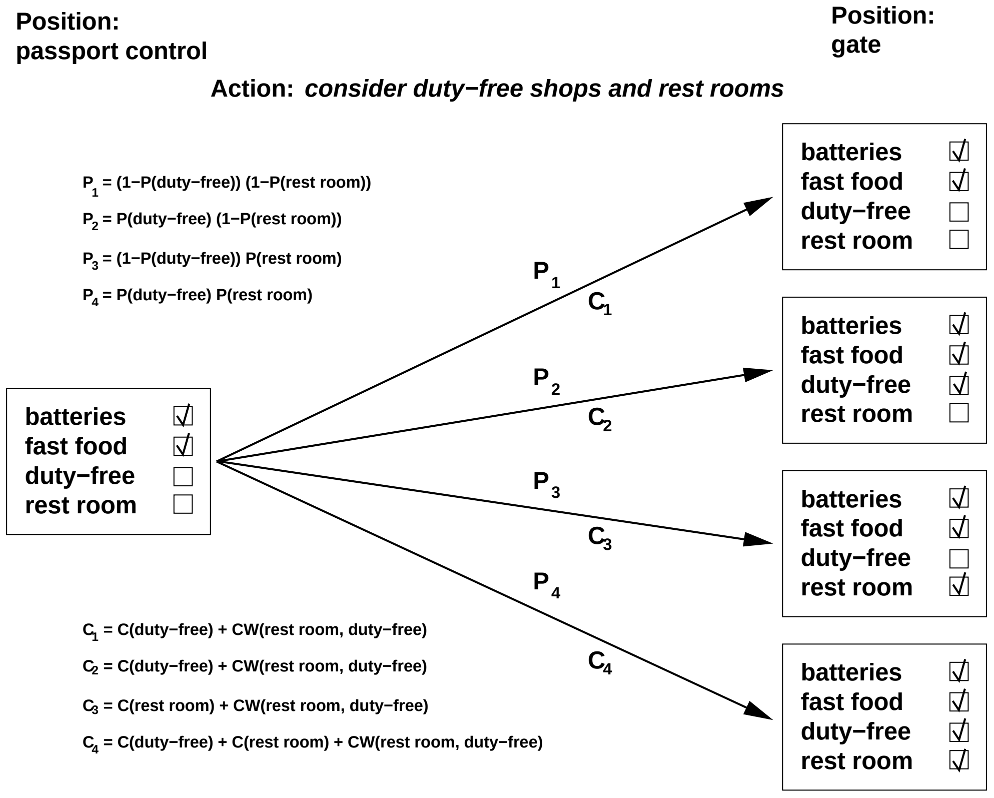

Position: passport control
Position: gate
Action: consider duty−free shops and rest rooms
P1 = (1−P(duty−free)) (1−P(rest room))
P2 = P(duty−free) (1−P(rest room))
P3 = (1−P(duty−free)) P(rest room)
P4 = P(duty−free) P(rest room)
batteries ☑ fast food ☑ duty−free ☐ rest room ☐
P1 C1
P2 C2
P3 C3
P4 C4
batteries ☑ fast food ☑ duty−free ☐ rest room ☐
batteries ☑ fast food ☑ duty−free ☑ rest room ☐
batteries ☑ fast food ☑ duty−free ☐ rest room ☑
batteries ☑ fast food ☑ duty−free ☑ rest room ☑
C1 = C(duty−free) + CW(rest room, duty−free)
C2 = C(duty−free) + CW(rest room, duty−free)
C3 = C(rest room) + CW(rest room, duty−free)
C4 = C(duty−free) + C(rest room) + CW(rest room, duty−free)

***Figure 5.3**: The macro-level action "consider duty-free shops and rest rooms as secondary goals".*
(Transition costs and probabilities of macro-level actions can be determined heuristically from costs and probabilities specifi ed for micro-level actions as explained in Section 5.3.1. Thereby, $P(X)$ corresponds to the probability to achieve secondary goal X together with the primary goal at hand, $C(X)$ corresponds to the costs associated with the cumulated micro-level costs—costs that correspond to distances from one infrared beacon to another—to achieve secondary goal X, and $CW(X_1, \ldots, X_n)$ corresponds to the costs to actually achieve secondary goals X_1 to X_n, for example, the average time needed to visit a duty-free shop and buy a duty-free item.)

For the state on the left-hand side of Figure 5.3, the decision-theoretic model includes the transition of three further actions: *consider (only) duty-free shops*, *consider (only) rest rooms*, and *do not consider any secondary goal at all*. The macro-level policy will always switch to the action *do not consider any secondary goal at all* for all states if the time to achieve a primary goal runs short. In other words, for some t, the t-time-units-to-go policy, which is computed as described in Section 4.4.1, will assign the action *do not consider any secondary goal at all* to all states.

The user might also achieve a goal that is actually not recommended to be considered on the way from the passport control to the gate. For example, he might visit a rest room on his own initiative, while the macro-level policy (and thereby the recommendations determined by the micro-level policy) take only the duty-free shops into account. To consider this in the decision-theoretic model, one defines the corresponding transitions, which are usually associated with small probabilities. That is, in the example at hand, one of the transitions for the action *consider duty-free shops* ends in a state for the position at the gate with the secondary goal of visiting a rest room achieved while the secondary goal of having a duty free item bought is not achieved.

Despite the small number of locations that are considered in the macro-level decision-theoretic planning, the state space can be large. It grows exponentially in the number of secondary goals that are considered. Because the number of locations considered on the macro-level is small, the problem remains tractable for a medium number (up to about 10) of secondary goals. A particular characteristic of the airport scenario is favorable to the complexity of the macro-level decision-theoretic model: As the order of the primary goals is fixed, the state features do not need to include information about which macro-level locations have already been visited or which primary goals have already been achieved, respectively. Appendix F lists a part of a sample trace of the computations for the macro-level decision-theoretic planning in the airport assistance scenario.

5.3.3 Decision-Theoretic Planning on the Micro-Level

Once the macro-level decision-theoretic planning has determined which secondary goals are supposed to be considered together with which primary goals, the micro-level decision-theoretic model can proceed very much like we have described it for a scenario of low complexity in Section 4.2.3. In the model described in Section 4.2.3, we have only considered one secondary goal together with a primary goal. However, extending this model in a way that several secondary goals can be considered involves only two steps: (1) For all secondary goals, the topology description (describing how adjacent locations are connected) is augmented with the information about possibilities to achieve the secondary goal on the way from one location to another, and (2) new state features are added to the state description in order to maintain the information about which of the secondary goals (that are considered with the primary goal at hand) have already been achieved. It depends on the user's current location and the set of already achieved secondary goals where the user is recommended to go to and whether he is made aware of possibilities to achieve any of the secondary goals (which he has not achieved yet) along the upcoming part of the way. The rewards for the secondary goals, which express the importance that the user assigns to each of them, are identical to those used in the macro-level decision-theoretic planning process.

An important advantage of decision-theoretic planning in general is that policies, as they consider all possible situations that can—as long as they are encoded in the decision-theoretic model—occur during the interaction of the user and the system, can be computed offline. If the micro-level policies are supposed to be computed offline in the hierarchical decomposition approach, then we have to compute a set of finite-horizon decision-theoretic policies for each macro-action that can be proposed by the macro-level policy. A macro-action can have different outcomes, that is, it is not known in advance which of the secondary goals that are considered with a particular primary goal will actually be achieved. For example, the macro-level policy might specify for the situation when the user is at the check-in counter at time t and has not yet achieved any secondary goal, to consider (1) visiting a fast food restaurant and buying batteries, (2) visiting a fast food restaurant only, (3)buying batteries only, or (4) not trying to achieve any secondary goal at all. In this case, four sets of micro-level policies have to be computed and the appropriate one has to be selected at runtime. In the prototypical implementation running on a workstation, we refrained from precomputing all possible micro-level policies (1) because there was enough computing power available to do the planning online, and (2) because online micro-level planning allowed to incorporate updated information from the user modeling component from time to time.

In Section 5.3.1, we have described how the time restriction for the achievement of each of the primary goals can be computed. The fixed deadlines are considered by finite-horizon decision-theoretic planning just as it is described in Section 4.4. However, on the micro-level, it is not sufficient to compute the t-time-units-to-go policies up to the maximal amount of time that is assigned to the achievement of a primary goal on the macro-level. As we have mentioned before, the user will usually not achieve a primary goal exactly in the maximal time that is assigned to this primary goal, but will need less time than he actually has available. This means that saving time while achieving one primary goal allows the user to spend more time to achieve later primary goals. Therefore, the planning horizons of the individual micro-level decision-theoretic planning processes must overlap in the sense that the sum of the planning horizons of the micro-level decision-theoretic planning processes is greater than the total available time.

To guarantee that the planning horizon is sufficient, one can simply define the total available time as the horizon for each individual micro-level decision-theoretic planning process. However, this strategy results in many t-time-units-to-go policies that will never be used. As a better heuristic for the maximal finite horizon required for the achievement of a particular primary goal on the micro-level, we subtracted the sum of the minimal times to achieve each of the preceding primary goals from the total available time in the implementation.

Alternatively, if the micro-level policies do not need to be computed in advance (i.e., if the computing power of the handheld device allows online decision-theoretic planning), then one can subtract from the total available time the actual time that has already elapsed during the achievement of preceding primary goals. If a computation of the micro-level policies on demand is possible, that is, if the hardware on which the system runs is sufficiently fast, then the computation on demand is also preferable because it requires only a single set of micro-level policies—instead of several—to be computed for the achievement of each primary goal. The reason is that the exact state from which the user starts to achieve a particular primary goal is already specified under these circumstances, that is, the system knows which secondary goals the user has already achieved together with preceding primary goals.

5.3.4 Integrating Macro- and Micro-Level Decision-Theoretic Planning

The integration of the macro- and micro-level decision-theoretic planning processes results from the alternating application of the macro-level policy and appropriate micro-level policies. Figure 5.3.4 summarizes the algorithm computing the micro-level policies on demand (i.e., when the micro-level policies are not precomputed).

After the macro-level policy has been computed, the system checks which secondary goals are supposed to be considered on the micro-level in the situation at hand. According to the unachieved secondary goals (initially, all secondary goals are unachieved), the system selects the t-time-units-to-go policy according to the remaining time and starts giving navigation recommendations to the user.

When the user achieves the first primary goal (i.e., when he reaches the check-in counter in the airport scenario), the system computes the set of finite-horizon policies for the next primary goal

```
HIERARCHICAL DTP($G^P, G^S, R^S, T^{micro}_{costs}, T^{micro}_{probs}, Topo, Dead^{macro}$)
$T^{macro}_{costs}$ ← determine_macro_action_costs($T^{micro}_{costs}, Topo$)
$T^{macro}_{probs}$ ← determine_macro_action_probs($T^{micro}_{probs}, Topo$)
$MDP^{macro}$ ← build_MDP($G^P, G^S, R^S, T^{macro}_{costs}, T^{macro}_{probs}, Topo$)
$Policy^{macro}$ ← value_iteration($MDP^{macro}, Dead^{macro}$)
while ¬all_primary_goals_achieved do
    $G^P_i$ ← next($G^P$)
    $G^S_i$ ← apply_policy($Policy^{macro}, Time$)
    $Dead^{micro}$ ← compute_micro_deadline($Dead^{macro}, Topo$)
    $MDP^{micro}$ ← build_MDP($G^P_i, G^S_i, R^S, T^{micro}_{costs}, T^{micro}_{probs}, Topo$)
    $Policy^{micro}$ ← value_iteration($MDP^{micro}, Dead^{micro}$)
    give_recommendation($Policy^{micro}, Time$)
od
```

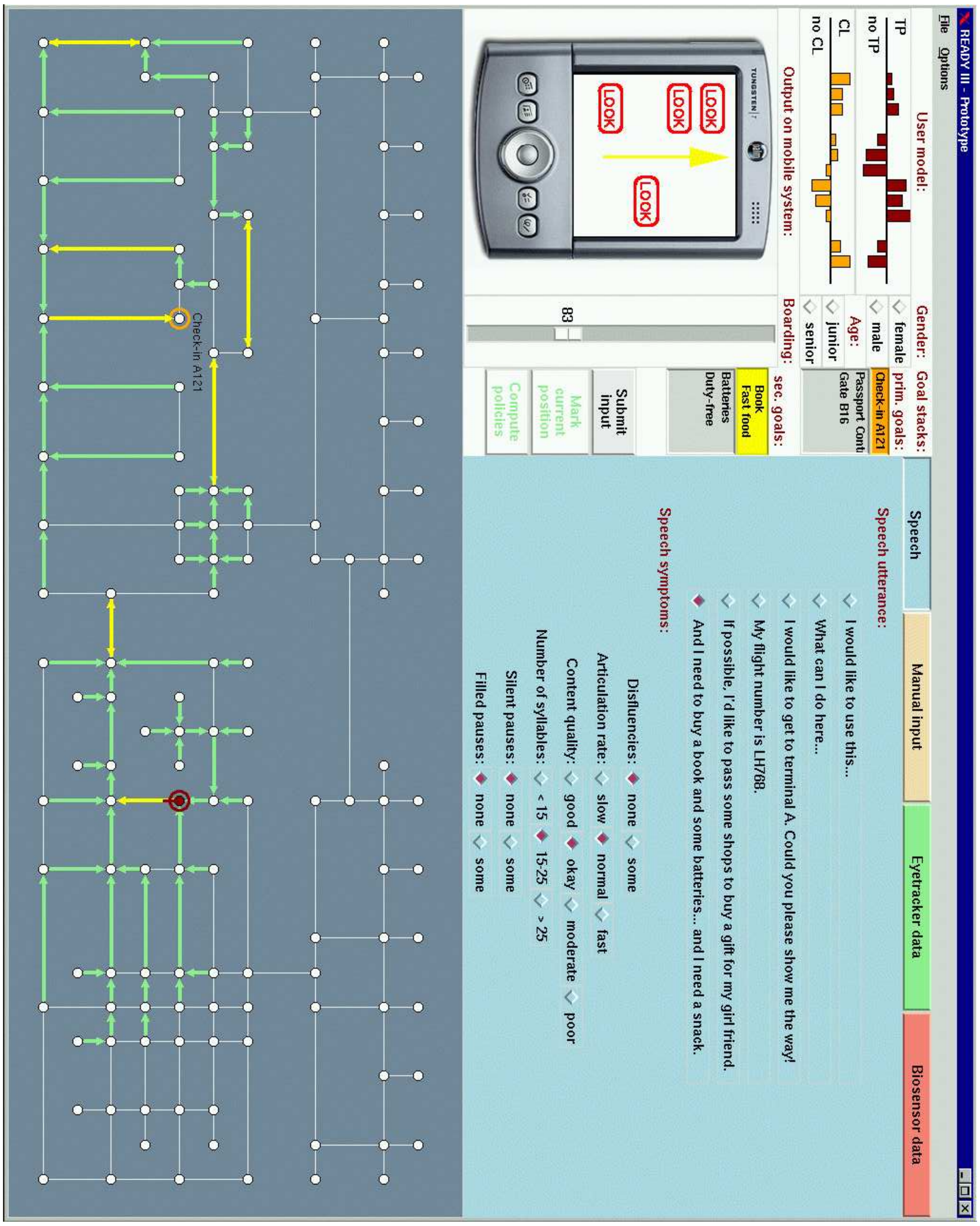
***Figure 5.4**: Hierarchical decision-theoretic planning.*
(G^P: list of primary goals; G^S: list of secondary goals; R^S: rewards for secondary goals; T^{micro}_{costs}: cost table for macro-level actions; T^{micro}_{probs}: probability table for macro-level actions; $Topo$: topology description, $Dead^{macro}$: overall deadline. Further explanations are given in the text.)

(e.g., getting to the passport control) according to which of the secondary goals have been achieved during the achievement of the preceding primary goal. From this set, the system selects the right t-time-units-to-go policy again and applies it by giving navigation recommendations to achieve the next primary goal. This procedure is repeated until all primary goals have been achieved (i.e., in the airport scenario, until the user has reached the gate).

We have implemented the hierarchical approach described in this section for the airport scenario prototypically on a workstation and not on a mobile device. Note that on a mobile device, the computation of the micro-level policies can only be accomplished on demand if the mobile hardware is fast enough. Under this assumption, on demand computation is in particular preferable if memory space is restricted. In fact, the system could be implemented such that it detects memory and micro-processor restrictions and then decides automatically whether to precompute the micro-level policies or to compute them on demand. This was a contribution to the system's resource-adaptivity on a different dimension.

5.3.5 System Behavior

The hierarchical decision-theoretic planning approach was implemented for the airport assistance domain. The user interface of the prototype (see Figure 5.5) includes controls to simulate the user input (upper right part of the figure) to a mobile airport assistance system, a visualization of the finite-horizon policies (lower half of the figure) used by the system, and eventually the simulated system output (upper left part of the figure). The screen shot captures the following simulated situation: The user has just entered one of the airport entrances and specified his overall goal with the speech utterance "My flight number is LH768." From this information, the system derived three primary goals by a database look-up: check-in B121, passport control B, and gate B16. Moreover, the system derived the time until the boarding for flight LH768. Asked if there is anything the system can try to help with, the user specified four secondary goals via further (simulated) speech utterances: "If possible, I would like to pass some shops to buy a gift for



Gender: female, male

Age: junior, senior

Boarding: 83

Goal stacks:

prim. goals: Check-in A121, Passport Contr, Gate B16

sec. goals: Book, Fast food, Batteries, Duty-free

Speech utterance:

- I would like to use this...
- What can I do here...
- I would like to get to terminal A. Could you please show me the way!
- My flight number is LH768.
- If possible, I'd like to pass some shops to buy a gift for my girl friend.
- [✓ And I need to buy a book and some batteries... and I need a snack.]

Speech symptoms:

Disfluencies:	[✓ none]	some		
Articulation rate:	slow	[✓ normal]	fast	
Content quality:	good	[✓ okay]	moderate	poor
Number of syllables:	< 15	[✓ 15-25]	> 25	
Silent pauses:	[✓ none]	some		
Filled pauses:	[✓ none]	some		

Figure 5.5: *Hierarchical, goal-priorized decision-theoretic planning in the READY prototype: Providing airport assistance.*
(Explanation in text.)

my girlfriend. And I need to buy a book and some batteries... and I need a snack." Primary goals, secondary goals, and the time until the boarding are displayed in the upper half of the user interface, left from the middle[7].

After the hierarchical decision-theoretic planning process has been triggered (via a button), the system first computed the macro policy. For the given remaining time, the macro policy specified that the first micro-level decision-theoretic planning process will consider two secondary goals with the primary goal *check-in B121*: buying a *book* and trying to find a *fast food restaurant*. The first micro-level policy considers only those areas in the airport terminal that are relevant to achieve one of the two secondary goals on the way to the check-in counter. The slightly darker (green) arrows in the policy visualization correspond to simple navigation recommendations (e.g., an arrow indicating the direction to go to) on the PDA display, the slightly lighter (yellow) arrows correspond to maps on the PDA display, which include a navigation recommendation and in addition information about opportunities to achieve the secondary goals that are currently considered (cf. Section 4.2.2).

The prototype allows to simulate the moving through the airport building using the cursor keys. At those places in the airport where secondary goals can be achieved (e.g., at a book store or a fast food restaurant), the user interface of the prototype allows to specify if the user actually achieves the corresponding secondary goal or not. This specification is realized by a pop-up window with check buttons. Moving through the airport building, the user notices how the time until the boarding decreases and the navigation recommendations and their presentation change, with the consequences that we have already described in Section 4.4.1: The less time is available, the more directly will the system try to guide the passenger to the individual navigation goals and the less importance will the system assign to making the user aware of opportunities to achieve any secondary goals along the way. After the user has reached the check-in counter, another micro-level decision-theoretic planning process for the next primary goal will be triggered according to the remaining time and the macro policy. If the user follows the system's instructions in the example correctly, he will eventually reach the gate in time and have the chance to achieve all specified secondary goals along the way.

The prototype demonstrating the goal-priorized decision-theoretic planning approach—like all other prototypical implementations mentioned in this thesis—has been realized with the Mozart programming system[8].

5.3.6 Positive Side Effects of Decomposition

Decomposition has two positive side effects: (1) It contributes to the system's usability and (2) it allows to consider partially observable information without explicitly modeling them in a POMDP.

Usability: Decomposition can increase the system's transparency—one of the important usability issues that we will discuss in Section 7.4.1. A smart user, who tries to achieve several primary and secondary goals in the airport scenario without the assistance of an airport assistance system, would possibly try to decompose the problem in a similar way as described in this chapter. He

[7]The user interface of the prototype provides displays to visualize further information from the user model that can be used to parameterize the decision-theoretic planning process (cf. Section 4.1.6.2 and Section 4.1.6.3). Moreover, it provides controls to specify further information that can be used as input for the user modeling component: a specifi cation of the user's manual input, eye-tracker data, and biosensor data.

[8]Mozart is documented at and can be downloaded from `http://www.mozart-oz.org/`.

would intuitively distinguish primary and secondary goals and intuitively decide which of the secondary goals to consider together with a particular primary goal. However, without the detailed knowledge about the environment (as it is modeled for the micro-level decision-theoretic planning processes), he would not be able to plan ahead his trip through the airport terminal as profoundly as the system does—based on the goal-priorized decision-theoretic planning approach.

We suppose that many passengers in an airport scenario decompose planning problem similar to those described in this chapter in the way that the described system does. The similarity of the user's intuitive strategy and the goal-priorized decision-theoretic planning approach should help the user understand the system behavior better than if the system does not decompose the problem. Due to the macro-level decision-theoretic planning, the system can explain what secondary goals are considered with the primary goal at hand and what other secondary goals are not. In addition, the number of secondary goals that are considered together with a particular primary goal is smaller than if all secondary goals are considered in a single decision-theoretic planning process. Finally, the information gained in the decomposition process can be exploited to generate sophisticated explanations for the system's recommendations (see Section 7.4). For example, if requested why a particular secondary goal is currently considered, the system can refer to the density of opportunities to achieve this particular secondary goal in the area around the user's location at hand. In general, the system can first try to use information from the macro-level decision-theoretic model to give an explanation for its recommendations and use information from the micro-level decision-theoretic model if a more detailed explanation is requested.

Partial Observability: In Sections 4.1.6 and 4.2.6, we have described how information of the user model, in particular information about cognitive load and time pressure of the user, can be exploited in a decision-theoretic planning process. Cognitive load or subjectively felt time pressure refer to mental states of the user and can therefore not be observed directly[9] but must be considered to be partially observable. We have explained in Section 4.1.6 how this partially observable information can be inferred from observations (such as symptoms in the interaction of the user with the system or biometric information) by using Bayesian networks.

In principle, it is possible to incorporate the information about symptoms in the interaction of the user with the system in a partially observable Markov decision process and abandon an explicit user modeling component. Cognitive load and time pressure would then be two partially observable state features whose value depends on observations about the behavior of and the biometric data about the user, respectively. However, we have indicated in Section 4.1.6 how sophisticated and complex the user model based on the Bayesian network approach is, and it would by far exceed the capabilities of today's most performing exact algorithms to solve POMDPs if one tried to cover all relations between the symptoms and the variables COGNITIVE LOAD and TIME PRESSURE.

The idea of a hybrid approach with a separate user modeling and decision-theoretic planning component yielded an intuitive system behavior in the scenarios of low complexity that we have discussed in Sections 4.1.6 and 4.2.6. Therefore, it is not only a question of feasibility when thinking about discarding a sophisticated user modeling component based on a Bayesian network approach in favor of a POMDP approach. Apart from complexity reason, there is also the advantage of a clear modularization when keeping the assessment of important aspects the current situation (i.e., the user modeling) and the finding of a strategy to adapt to the identified situation (i.e., the

[9]Even if neuro scientists might at some point be able to measure mental states reliably, such methods are not likely to be incorporated in devices of everyday use in the near future.

decision-theoretic planning) in separate processes. Modularization facilitates the implementation and maintenance of the system.

The goal-priorized decision-theoretic planning approach introduced is this chapter allows to incorporate important partially observable information on the micro-level in an appropriate way. We do not consider information about the user's cognitive load and subjectively felt time pressure to be relevant in the macro-level decision-theoretic planning process. Cognitive load or the feeling to be under time pressure may last for some seconds or minutes but are not likely to be constant over the whole period of time that the user spends in the airport building. The goal-priorized decision-theoretic planning approach offers the opportunity to consider changes in the user model with each computation of a micro-level policy. Updated information about cognitive load and time pressure can be used to parameterize the micro-level decision-theoretic planning processes as described in Section 4.1.6.3. Once again, note that this requires the micro-level policies to be computed online, which is currently feasible on a powerful workstation but not on standard handheld computers.

5.4 Synopsis

In this chapter, we have introduced goal-priorized decision-theoretic planning (GPDTP), a new type of hierarchical decision-theoretic planning to deal with complex scenarios. In contrast to existing decomposition approaches, goal-priorized decision-theoretic planning takes advantage of two typical characteristics of complex assistance tasks that occur in domains such as the airport assistance domain: (1) the distinguishability of primary, secondary, and in general n-ary goals and (2) the existence of a fixed deadline for the overall task.

For the scenario at hand, we employ a two-level goal-priorization, which implies hierarchical decision-theoretic planning on two levels: (1) A macro-level decision-theoretic planning process determines which secondary goals to consider together with each individual primary goal and (2) several micro-level decision-theoretic planning processes solve the resulting low-complexity sub-problems in the way described in Chapter 4. Each of the resulting micro-level policies is used to achieve only a subset of all secondary goals together with an individual primary goal and considers only particular areas of the physical environment. The time restrictions for the complete task are propagated to the achievement of the individual primary goals in a way that the successive execution of the micro-level policies suffices the time restrictions for the complete task. We will consider further aspects of (a broader) goal ranking in Chapter 6.

The diminution of the planning horizon in the micro-level decision-theoretic planning processes (compared to the time horizon of the complete process) and the strategy to determine some of the parameters for the macro-level decision-theoretic planning process heuristically (e.g., the assignment of a particular portion of time for the achievement of an individual primary goal) mean that the goal-priorized decision-theoretic planning approach can only be an approximation. However, a single (i.e., not decomposed) decision-theoretical model, which can treat primary and secondary goals in a similar way as our decomposition approach does, is too complex for exact decision-theoretic planning approaches. It grows exponentially in the number of primary and secondary goals because the information about the achievement of all goals has to be included in each state.

The hierarchical decomposition approach that we have introduced is applicable only to problems that allow at least a distinction of primary and secondary goals. The existence of a fixed deadline for the overall task is helpful for the decomposition approach but not obligatory. The approach could without much effort be adjusted to a situation without time restrictions. The key idea that leads to a reduction of the problem complexity is to guarantee that the system is not obliged to consider all secondary goals in all micro-level decision-theoretic planning processes. If the number of considered secondary goals is small in the micro-level decision-theoretic planning processes, then they can be computed in reasonable time even though the micro-level state spaces grow exponentially in the number of secondary goals.

Hierarchical decomposition assuming a fixed order for the achievement of primary goals makes the macro-level state space grow only linear in the number of primary goals. Further ease of the complexity problem is achieved by considering in the macro-level decision-theoretic planning process only those physical locations that are associated with a primary goal. This way, also the macro-level policy can be computed in reasonable time as long as the number of secondary goals is not too large.

All decomposition approaches that we have surveyed at the beginning of the chapter do only consider one type of goals—those that correspond to what we call primary goals. Apart from complexity reasons, making the distinction between primary and secondary goals in the airport assistance domain allows for a problem decomposition that contributes to a more transparent system behavior and thereby increases the user's understanding of the system's decision making. Transparency improves the system's usability. Eventually, the goal-priorized decision-theoretic planning approach allows to consider at least some partially observable information in the micro-level decision-theoretic planning processes without requiring to switch to a POMDP model with discouraging complexity.

6 Applying DTP for User-Adaptive Systems: Workflow

Preceding chapters have only indicated how to get from a user-adaptive system planning problem to an appropriate decision-theoretic model and a good solution to the problem. The aim of the current chapter is to make this procedure comprehensible on a more abstract level, such that it can be applied to a range of user-adaptive system planning problems similar to those considered in this thesis. We will first describe the workflow for the development and application of decision-theoretic planning in the context of user-adaptive system problems of low complexity, i.e., for problems similar to those that we have described in Chapter 4, and then extend the workflow description so that it covers the application of decision-theoretic planning in the context of more complex scenarios, such as the example discussed in Chapter 5. We illustrate the development and application of a decision-theoretic model in the context of user-adaptive systems by referring back to the examples of the preceding chapters. At the end of the chapter, we will survey some approaches acquiring the necessary quantitative measures needed for the decision-theoretic approach.

6.1 Workfbw Overview: Scenarios of low Complexity

This section provides an overview of the workflow for applying decision-theoretic planning for user-adaptive system problems of low complexity. The basic concepts (i.e., the elements of Markov decision processes and the algorithms to solve them) that are mentioned in Figure 6.1 have already been explained in Chapter 3. Before we describe in detail the individual development steps, we first illustrate the chronology of the development process and the relationship between a user-adaptive system problem specification and the basic decision-theoretic planning concepts. Some of these interrelations might become clear to the reader only in the next section, when we illustrate the workflow with some examples from preceding chapters.

Figure 6.1 describes the individual steps that are necessary for the development and application of a decision-theoretic model for a scenario with low complexity. The figure indicates the chronological order in which the individual steps should ideally be taken. However, for some steps, there is no strict order that the system developer needs to stick to.

The software developer starts by *checking the applicability of the decision-theoretic planning approach*. Some processes in the context of user-adaptive systems, which can be planned in principle, might not be appropriate for a decision-theoretic planning approach. To find out about the appropriateness of the decision-theoretic planning approach, the developer answers some ques-

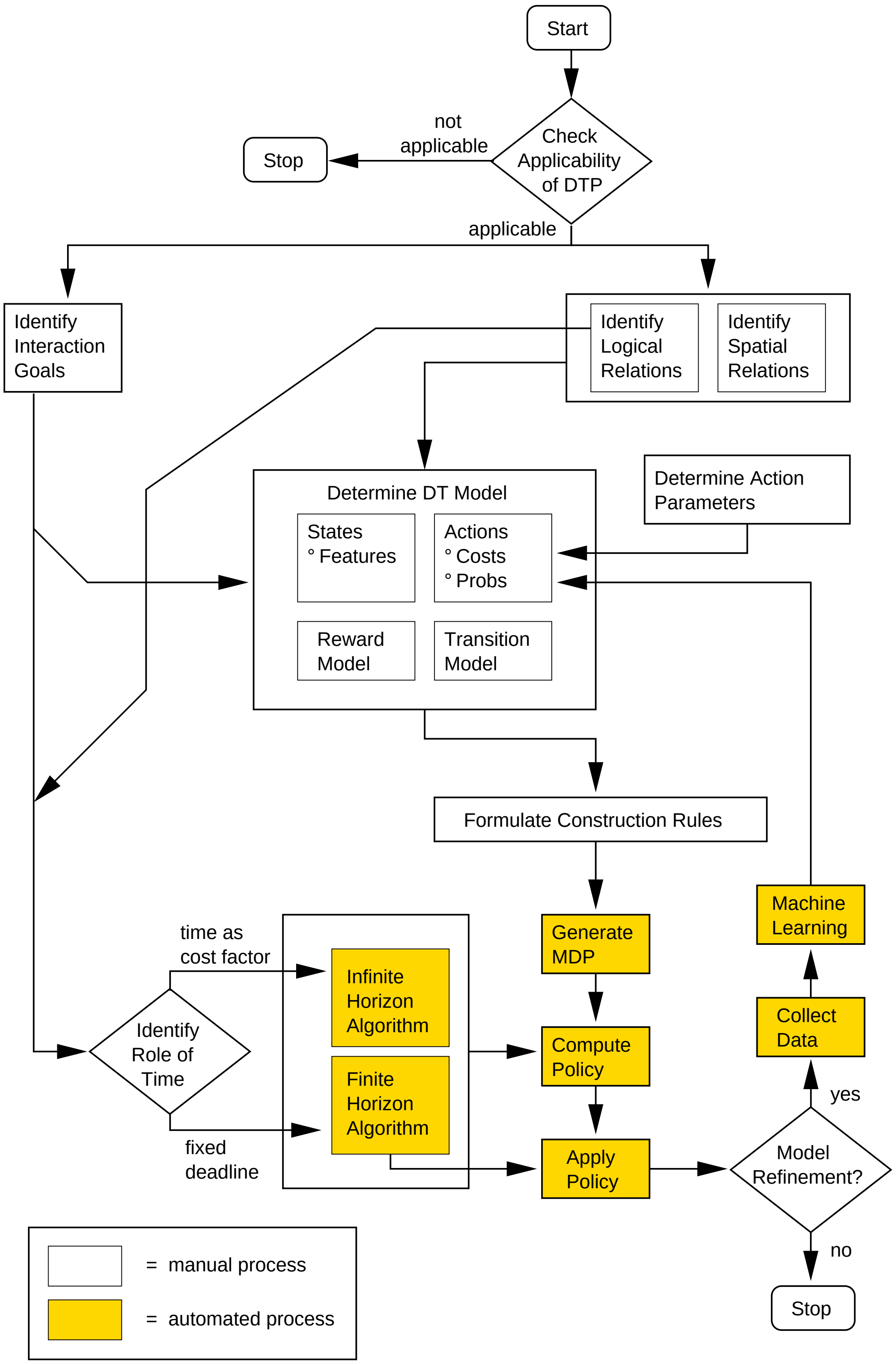


***Figure 6.1**: Workflow for the application of DTP in user-adaptive systems.*
(Explanation in text.)

tions that shed light on the nature of the interaction that is to be planned.

If the result is positive, then the next step is the identification of important top level characteristics of the interaction. In the figure, we distinguish the *identification of interaction goals* and the *identification of logical and spatial relations*. Decision-theoretic planning can consider goals to be obligatory or optional. The identification of the interaction goals determines not only the basic elements of the Markov decision process (such as the required state features or the reward function) but also if an infinite or finite planning horizon is appropriate for the problem at hand. The *identification of logical relations* between the actions that make up the interaction to be planned (e.g., the order in which two actions have to be executed) determines in particular the transition function. Some problems additionally exhibit spatial structure. The *identification of spatial relations* (e.g., the connections between individual locations or what action can be executed at a particular location) is often straightforward.

From the identification of interaction goals, logical relations, and spatial relations, the developer should be able to map the interaction to be planned to an initial decision-theoretic model by *determining states, actions, the reward model, and the transitions model*. There is usually not just one correct decision-theoretic model to describes the process to be planned. The determination of a “good” decision-theoretic model (i.e., one that describes the dynamics of the process sufficiently well and not unnecessarily verbose) is a process that requires some creativity by the developer. Eventually, it requires the *determination of action parameters* (costs and probabilities), which can be gained, for example, by expert judgments or experiments.

In principle, the decision-theoretic model describes the complete Markov decision process. However, if we want to build the Markov decision process explicitly as described in Section 3.1.5.2, a declarative description of the MDP is not sufficient. It is necessary to *formulate construction rules* to generate the required object-oriented state-transition diagram. As long as no general interpreter for construction rules exists, the construction rules have to be encoded in the programming language chosen for the project. The formulation of the construction rules is made easier if it is based on a “good” decision-theoretic model (see above). Once the construction rules are formulated, the *generation of the Markov decision process* is automatic.

Before the *computation of a policy*, it has to be clarified if the problem must be considered as an infinite-horizon or a finite-horizon problem. If time can be considered as a pure cost factor, the policy is computed with an *infinite-horizon algorithm*; if a fixed deadline has to be considered, the policy is computed with a *finite-horizon algorithm*. The *identification of the role of time* can be derived from the nature of the interaction goals and the logical relations of events that make up the interaction.

The policy is the solution of the user-adaptive system planning problem, and the *application of the policy* allows the system to control the interaction with the user. If the quantities used for the initial decision-theoretic model were only estimates, it is possible to *collect data* during the use of the system and adjust the parameters (costs and probabilities) in the decision-theoretic model.

6.2 Development of a Simple Decision-Theoretic Model

We now go into some more detail of the individual development steps described in the overview. To explain and illustrate the individual steps of the workflow, we will refer back to the examples used in Section 4.1 and 4.2 where appropriate. We refer to these examples as the *phone assistance example* and the *simple airport navigation assistance example*.

Check Applicability of DTP Approach At the beginning of the development process, the developer has to answer the question: Can the problem at hand—a particular interaction between a system and its user—be addressed by decision-theoretic planning at all? The following questions can help the developer to decide if it is worthwhile to apply decision-theoretic planning to the problem at hand.

- *Is the interaction of a sequential nature?* In addition to long sequential interactions, decision-theoretic planning is also well suited to very small sequences of interactions (see the simple instruction problem in Section 3.1.1); however, if only a single decision needs to be made then other techniques are more appropriate (cf. Section 2.1).

- *Do system and user act alternately?* Acting alternately does not mean that the system and the user each execute exactly one action in turn; one turn can consist of several actions by either the system or the user (e.g., in the simple instruction problem of Section 3.1.1 and in the phone assistance example of Section 4.1, both the system and the user can execute several actions in succession before the turn changes.)

- *Will the system have to trade-off multiple goals in the interaction with the user?* The consideration of competing interaction goals makes the decision-theoretic modeling particularly interesting; in the phone assistance example, the system trades off completing the interaction with the user fast and completing the interaction without the user making a mistake; in the simple airport navigation assistance example, the system trades off helping the user to reach the gate fast and helping the user to buy a gift along the way to the gate.

- *Is the outcome of the user's (or the system's) actions uncertain?* Although it is possible to consider completely deterministic processes by decision-theoretic planning, the strengths of DTP are not brought out this way.

- *Can the feedback that the system perceives about the progression of the interaction be considered to be reliable (i.e., is the interaction process fully observable)?* If the process is completely unobservable or only partially observable, other methods than those described in this thesis must be applied.

If the test about the reasonable application of decision-theoretic planning for the problem at hand is positive, then the next step of the development process is the identification of important top level characteristics of the interaction.

Identify Interaction Goals As the preceding chapters have illustrated, decision-theoretic planning can distinguish different types of goals, which have to be treated in different ways. Some goals are obligatory, for example, the completion of a dialog that includes executing a number of instructions or getting to a particular location after following a number of navigation recommendations. We have referred to obligatory goals as *primary goals*. In a simple decision-theoretic model, we can only deal with a single primary goal.

Some goals are optional, for example, accomplishing subordinated tasks along the way while following navigation instructions to a particular location. We have referred to optional goals as *secondary goals*. While not reaching a primary goal is associated with not receiving any positive reward at all, not reaching a secondary goal (while following a particular primary goal) only decreases the reward for the primary goal. In a simple decision-theoretic model, one or few secondary goals can be considered.

The identification of the interaction goals generates the question: What propositions must hold in a goal state? Answering this question will elicit characteristics of the state space (in terms of relevant state features) in general, as well as characteristics of potential actions (in terms of transition properties). Moreover, the identification of the interaction goals is a prerequisite for the determination of a reward model.

Identify Logical and Spatial Relations Some user-adaptive system problems do not involve considering any spatial structure at all (e.g., the simple instruction problem that we have introduced in Section 3.1.1 or the phone assistance problem. For such problems, only the logical relations of the system's and the user's actions have to be identified. While there are user-adaptive system problems that lack any spatial structure, all interactions of a system with its user that we address with decision-theoretic planning must exhibit some logical structure.

Identifying logical relations corresponds to answering questions like, for example: How does an action change the world state? What is the duration of an action? In what state can the system take an action? In what state can the user take an action? Which action (of the system/of the user) can be followed by which other action (of the system/of the user)?

Identifying spatial relations corresponds to answering questions like, for example: How can spatial information (e.g., a location at which the user receives a recommendation by the system) be encoded in the state description? How is the user's movement in a physical environment reflected in the the state transitions? How is the spatial information interweaved with other state information, for example, with the information about where the user can achieve a secondary goal dependent on the recommendations of a system?

The identification of logical and spatial relations of states and actions that describe the interaction between a system and its user is usually not done in isolation. It is rather accomplished in parallel with the identification of the interaction goals. Identifying logical or spatial relations within the interaction can give hints for the identification of interaction goals and vice versa.

Determine Elements of MDP Once the interaction goals as well as logical and spatial relations of the user-adaptive system problem are identified, the developer can start to define the elements of a Markov decision process for the problem at hand. It is reasonable to determine first a set of state features that appears to be relevant for the process to be planned[1]. State features do not need to be boolean in general—as it is sometimes assumed for practical reasons when a factored representation is involved—but they should be kept simple. For example, the working memory content of a user in the simple instruction problem (cf. Section 3.1.1) can be described sufficiently in terms of two simple (integer valued) features (N IN WM and N IN BUNDLE) instead of using a list that describes the working memory content unnecessarily in greater detail. Similarly, locations can be represented by a simple integer state feature for the location ID. For some state information, boolean state feature are sufficient, for example, to describe if the user has achieved a particular secondary goal or not.

[1]From our own experience, we can report that the initially identifi ed state features sometimes have to be adjusted or refi ned later on in the development process, for example, because it turns out that some characteristics of the process can be described more compactly or more comprehensible with slightly different state features.

After the relevant state features have been identified, the next step is the determination of the available actions and the transition model. Actions can often be described on an abstract level and thereby be generalized in the transition model. For example, in the simple instruction problem, the decision-theoretic model takes only two (generalized) actions into account: GIVE INSTRUCTION—the system has the initiative—and WAIT FOR EXECUTION—the user has the initiative. It is not necessary to specify an action in more detail than is really needed for a concise model: In the decision-theoretic model for the simple instruction problem, it does not matter if the instruction that the system gives (or the user executes) is, for example, *set B to 2* or *set J to 4*. Important for the decision-theoretic model is only the information about how long the system needs to give and how long the user needs to execute the corresponding instruction, and what the probability is that the user executes the instruction correctly. For example, if the execution of *set B to 2* is by itself more error prone than the execution of *set J to 4*, then this difference has to be considered in the decision-theoretic model—which in fact can be the case if actions are not as homogenous as in the simple instruction example. In the simple instruction example, we have assumed that the probability of a user executing an instruction correctly depends only on the bundling size and the number of instructions that the user has in his working memory when he executes an instruction—the same holds for the time needed to execute an instruction. Based on this assumption, it is possible to represent similar actions generically in the decision-theoretic model. The generic specification of actions is essential to create a Markov decision process from a concise set of construction rules (see below).

The transition model describes the effects that actions have on the state in which they are executed and thereby defines the structure of the Markov decision process for the problem at hand. It can be represented graphically: either on a general level (see, e.g., Figure 3.3) or for a particular (but sufficiently small) instance of a problem (see, e.g., Figure 3.4). A small instance of a transition model can help to define the general version. The general transition model is more useful for the formulation of the construction rules than the transition model for a particular instance. If the planning problem has a spatial component, the spatial structure can serve as a basis for the transition model.

The reward model follows from the identified goals. For example, having completed a sequence of instructions or reaching a particular location after following a sequence of navigation recommendations can be associated with a reward. The developer has some freedom with regard to the absolute value for reaching a goal. In general, the reward reflects the importance of reaching a particular goal and is traded off in the decision-theoretic planning process against the cost for the actions to reach the goal. Insofar, rewards and costs have to be coordinated.

Determine Action Parameters The need to collect (sometimes a large number of) numerical values for a Markov decision process (probabilities, cost, and rewards) is routinely criticized as a form of the *knowledge acquisition bottleneck*[2] of decision-theoretic planning. Gaining precise numbers is in fact associated with considerable effort. In Section 3.1.1, we have described one way to deal with this problem: an experiment that allowed us to derive costs and probabilities for a decision-theoretic model that describes the dynamics of the experimental setting sufficiently well.

[2]The knowledge acquisition bottleneck was originally formulated in the context of expert systems; machine learning as a general approach to overcome the knowledge acquisition bottleneck is discussed, for example, by Alonso, Mat´e, Juristo, Muñoz, & Pazos, 1994; for Bayesian networks, which similar to MDPs require the collection of large amounts of probabilities, Skaanning, 2000, has proposed a knowledge acquisition tool to overcome the knowledge acquisition bottleneck.

Another warrantable approach, for example, is the collection of judgments by experts. As this topic is of particular importance, we go into more detail about the acquisition of of the numerical values in Section 6.4.

Formulate Construction Rules We have mentioned above that the formulation of construction rules takes the transition model as a starting point. Thereby, it is easier to revert to a general transition model to formulate construction rules than to derive them from a particular instance of a problem. Insofar, the specification of a general transition model is recommendable before formulating the construction rules.

The construction rules can be considered to be (mainly) if-then-rules, which describe the complete construction of a Markov decision process. Consider the simple instruction problem. As an example, we formulate a subset of the rules to build the structure described in Figure 3.4.

- Start by creating a state with N TO GIVE $= 2$, N IN BUNDLE $= 0$, N IN WM $= 0$, and CORRECT $= true$ (where $true$ corresponds to $+$ in the figure).
- If N TO GIVE $\neq 0$ and N IN BUNDLE $=$ N IN WM, then create a transition for the action GIVE INSTRUCTION and the successor state with N TO GIVE' $=$ N TO GIVE -1.
- If N IN WM $\neq 0$, then create two transitions for the action WAIT FOR EXECUTION and the successor states with CORRECT $= true$ and CORRECT $= false$ (+ and – in the corresponding Figure 3.4, respectively), N IN WM' $=$ N IN WM -1, and N IN BUNDLE' $= 0$ if N IN WM' = 0.

We have simplified the construction rules for this example. According to our experience, a large part of the effort for an object-oriented implementation of a Markov decision process is usually dedicated to the formulation of appropriate construction rules.

An interesting line of research in this context is followed by Boutilier, Reiter, Soutchanski, and Thrun (2000) and Boutilier, Reiter, and Price (2001), who present a framework in which Markov decision processes can be specified in first-order logic and that permits a symbolic representation of value functions and policies.

Generate MDP Once the construction rules have been formulated, the *generation of the Markov decision process* is automatic. If the problem at hand exhibits some spatial structure, the construction process will be geared to this spatial structure.

In the simple airport navigation assistance example, the construction procedure can, for example, first generate two *state objects*—one representing the state where the user has not yet bought a gift, one representing the state where he has—and then connect the state objects according to the specification of the state transitions. That is, the construction procedure checks for each particular state what other states can be reached from that state and maintains a pointer to them in the corresponding state object, as well as the information about costs and probabilities related to the corresponding transition[3].

For the phone assistance example, the generation of the MDP is slightly more complex. As there is one particular start state, it is more appropriate to create the corresponding state object first

[3]We describe the strategy that has shown to work well in the implementation; however, other strategies might work equally well, for example, alternately generating state objects and transitions, as we have done for the phone assistance example.

and then start a recursive procedure that creates all possible successor state objects and connects them (via pointers) according to the generation rules. This results in a *depth-first MDP construction process*. The leaf state objects of such an object-oriented state diagram correspond to the goal states of the Markov decision process.

Identify Role of Time We distinguish two roles of time for a decision-theoretic planning process: (1) Time as a pure cost factor, such that it can be traded off against the reward for reaching a goal state; and (2) time as a fixed quantity, such that it additionally constitutes a fixed deadline for the end of the process to be planned.

If for an interaction planning problem, there is no fixed point in time when a primary goal has to be reached (but it is better if the primary goal is reached earlier than later), then it is reasonable to consider time as pure cost factor for that problem. If there exist one or several secondary goals, the consideration of time as cost factor allows to trade off the costs for actions that help to achieve a secondary goal against the reward for achieving the primary goal. An *infinite-horizon algorithm* will be appropriate to solve a problem of this kind. Later on, during the interaction of the system with its user, the same policy will be valid throughout the complete process, that is, until the primary goal will eventually be achieved.

Where one or several secondary goals are involved and the time available to achieve a primary goal is limited by a fixed deadline, the policy for the interaction planning problem will not be the same for different points in time. The more the fixed deadline approaches, the less time consuming actions helping the user to achieve a secondary goal can the system select. In the simple airport navigation assistance example, we have seen how the system selects quicker routes and gives less information about opportunities to buy something along the way to the gate as the deadline—in this case, the boarding—approaches. A *finite-horizon algorithm* will be appropriate to solve a problem of this kind. Later on, during the interaction of the system with its user, the system will have to apply the right t-time-units-to-go policy (cf. Section 4.4.1) according to the time remaining until the fixed deadline.

A more complex model has to be applied if the costs for transitions or the rewards for goal states vary over time. The TMDP approach, which we have described in Section 4.4.2, is appropriate for problems of this kind. We have not considered problems in the context of user-adaptive systems that require this kind of time-dependent decision-theoretic planning.

Compute Policy In Section 3.1.5.2, we have described the three phases of the computation of a policy (i.e., the application of value iteration) on an object-oriented representation of a state-transition diagram: (1) the initialization of the state values, in particular the values of the goal states, (2) the simultaneous computation of the value function for the next stage, and (3) the simultaneous update of the values of all states. The three phases of the computation are independent of the structure of a Markov decision process, that is, we can apply them to the decision-theoretic model of the phone assistance example and the decision-theoretic model of the simple airport navigation assistance example equally well.

According to the role of time for the problem at hand, step (2) and (3) of the algorithm will be repeated for a fixed number of iterations (in the finite-horizon case, i.e., where a fixed deadline is involved) or will be repeated until another iteration yields no further alteration of the policy anymore (cf. the paragraph regarding the stopping criterion in Section 3.1.4.2).

Apply Policy The application of a policy is simple. As a policy is a mapping from states to actions (cf. Section 3.1.4), it can be represented by a look-up table that can easily be installed and applied on any handheld device. Applying a policy means merely perceiving the current state and finding the optimal action for that state in the policy (look-up table).

One important advantage of controlling the interaction of a system with its user based on a policy has already been mentioned in Section 5.3.3: The execution of a policy that is completely computed offline needs hardly any computational resources at run time, that is, the system can react immediately as soon as a new state is perceived. This advantage may not be surprising, as it comes along with the (moderate[4]) disadvantage of the memory space consumption for representing the policy. In this sense, a policy reflects the classical trade-off between time and space in computer science.

Model Refinement? If it was not possible to determine the parameters of the decision-theoretic model precisely beforehand, the system developer has the opportunity to adjust the decision-theoretic model taking collected data during the first application into account.

Collect Data/Machine Learning The collection of data for the adjustment of the decision-theoretic model used for the problem at hand is not obligatory. Sometimes, the costs and probabilities used in a decision-theoretic model can be determined reasonably accurate prior to the application of the policy in practice. In cases that do not allow to acquire these data with adequate precision in advance, an initial model can be based on estimations and later—once the system is put into practice—be refined by adjusting costs and probabilities according to data gained during first applications of the policy based on the initial model. If the parameters of the initial model are only estimates, the initial policy will in general be suboptimal but can be considered as a more or less (dependent on the accuracy of the estimated parameters) good approximation of the optimal policy. Machine learning techniques can be applied to adjust the decision-theoretic model. We will discuss the acquisition of the parameters (in particular action costs and the probabilities of action outcomes) for a decision-theoretic model in more detail in Section 6.4.

6.3 Development of a Hierarchical Decision-Theoretic Model

Figure 6.2 describes the necessary extensions of the workflow for the development and application of a goal-priorized decision-theoretic model. Compared to Figure 6.1, those parts of the workflow description that are new or extended are highlighted. The extension involves in particular the macro-level decision-theoretic planning and its preliminary computations, as well as the interleaving of the macro-level policy with micro-level policies.

Goal Priorization: Identify Goal Ranking For the application of the goal-priorized decomposition approach as we have described it in Chapter 5, apart from identifying several secondary goals, it is necessary to identify *several* primary goals. The example of Chapter 5 required only a simple goal priorization distinguishing primary and secondary goals. In principle, that is, if the considered planning process is complex enough, we can consider n-ary goals. Think of the

[4]The memory capacity of prevalent handheld computers is already far beyond what is needed to store the policy of any of the examples given in this thesis. However, sometimes a large portion of the available memory might already be occupied by other programs.

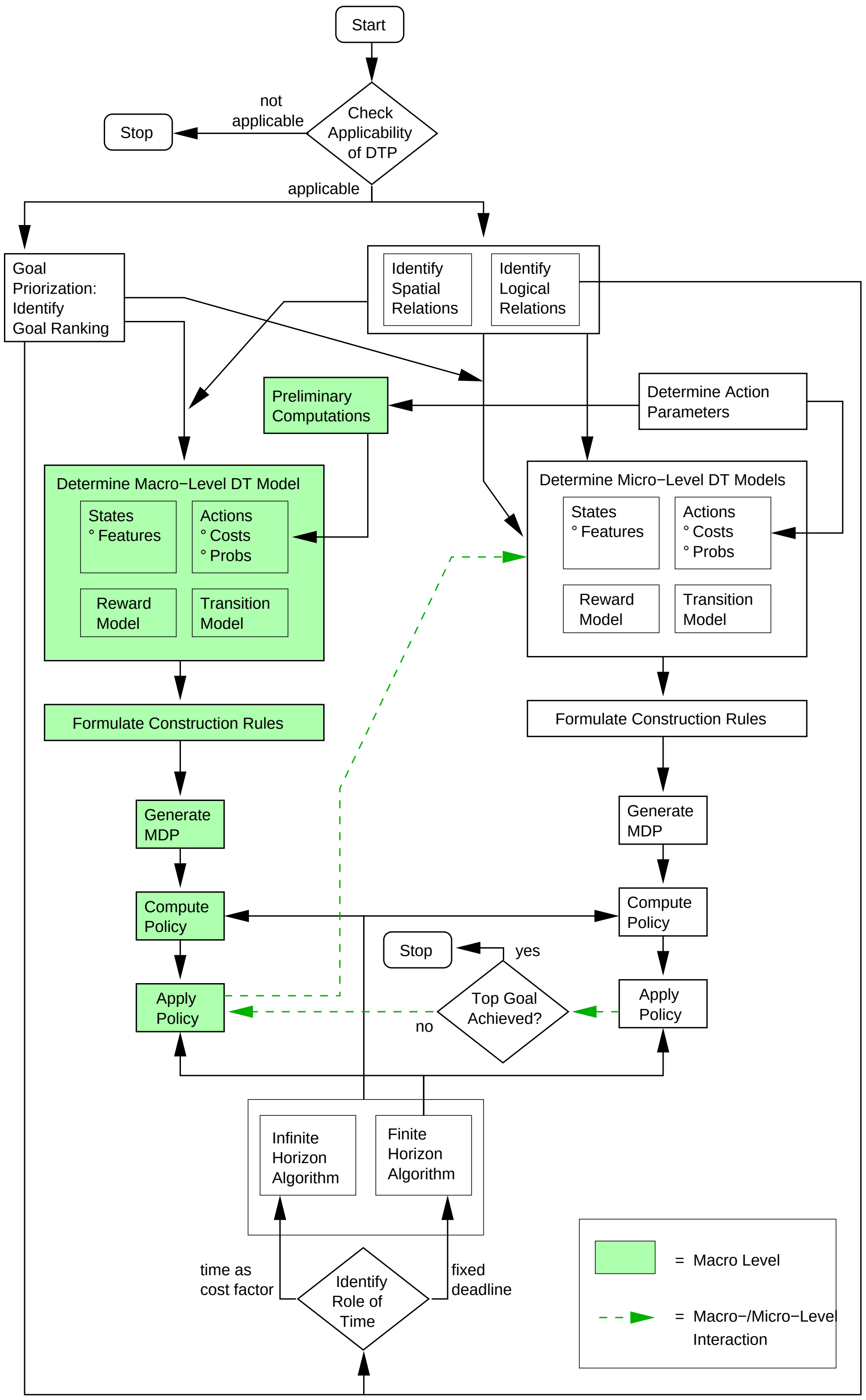


Figure 6.2: *Workflow for the application of GPDTP in user-adaptive systems. (Explanation in text.)*

following extension of the example given in Chapter 5: A system that guides the user from the city center to the airport. On the way, the user might want to pass by his office and visit a friend in hospital. He might want to buy some flowers on the way to the hospital and refuel his car on the way to the airport. In this scenario, we need additional goal ranks for the navigation between buildings and the given subordinated goals. However, there will be a natural bound for the scale of the goal ranking. The scenario might be extended to several cities, and also within the level that we have mentioned, the goal rankings might be refined a bit further. But in practice, the goal ranking will not exceed a medium number of levels.

In the airport assistance example of Chapter 5, primary goals are associated with particular locations (check-in counter, passport control, and gate). Consequentially, we can treat the information about the achievement of the primary goals in the macro-level decision-theoretic model in analogy to the location information in the micro-level decision-theoretic model. The order in which the primary goals are supposed to be achieved will be reflected in the structure of the macro-level Markov decision process.

Preliminary Computations In Section 5.3.1, we have explained the necessary preliminary computations for the macro-level decision-theoretic planning. These computations are based on the *logical and spatial relations* identified for the user-adaptive system problem at hand and take the *action parameters* determined for the micro-level decision-theoretic planning processes as input. For example, the computation of macro-level action costs and probabilities depends on the location associated with the corresponding primary goal and the availability of opportunities to achieve particular secondary goals in the adjacent area.

Macro-Level Decision-Theoretic Planning Some of the workflow steps for the development of a hierarchical decision-theoretic model are identical in the macro-level and the micro-level decision-theoretic planning: The macro-level development steps *determine macro-level DT model*, *formulate construction rules*, *generate MDP*, and *compute policy* can be accomplished in analogy to the corresponding steps on the micro-level (see Section 6.2).

Interleaving Macro-Level and Micro-Level DTP The interleaving of macro-level and micro-level decision-theoretic planning is indicated with two dashed arrows in Figure 6.2. The macro-level policy and the micro-level policies are executed alternately. Each action that the macro-level policy recommends specifies the secondary goals that are considered in the subsequent micro-level decision-theoretic planning process. If a micro-level policy has been executed, the system knows which secondary goals where achieved in parallel with the last primary goal and how much time the user needed to achieve the last primary goal. This information completes the system's knowledge about the current macro-level state. The system can determine the next macro-level action and start the corresponding micro-level decision-theoretic planning process thereafter. For each micro-level decision-process, it is possible to incorporate updated information from a user modeling component (see Section 5.3.6). The process terminates when the application of a micro-level policy results in a goal state that at the same time corresponds to a macro-level goal state. In the airport assistance scenario, this is the case when the passenger reaches the gate.

If more than two decision-theoretic planning levels are involved, that is, if the goal-priorization yields n-ary goals with $n > 2$, then the interleaving of the decision-planning processes from different levels can get more complicated and will in general not be domain independent.

6.4 Parameter Determination

For the decision-theoretic models that we have developed in the preceding chapters, a great many specific quantitative parameters need to be specified. These include measures like the duration of giving and following some abstract instructions, the probability of correctly following an instruction for operating a credit card phone, and the probability of finding, for example, a gift in a given store, or walking and waiting times in a shopping mall (see Chapter 7).

In contrast to reinforcement learning approaches, decision-theoretic planning requires some quantification of probabilities, costs, and rewards in an initial model. Because small changes of parameters in the decision-theoretic model do often not change the resulting policy, a system using a decision-theoretic model can sometimes yield reasonable results even without being based on parameters that reflect the processes in the real world precisely (cf. Section 4.2.5). Sometimes good parameter estimations are sufficient for the initial application of decision-theoretic planning. Moreover, even if a system applies a policy that is slightly suboptimal in some situations, it might still help the user manage the overall problem better than without any assistance at all. Therefore, it is often appropriate to start with inaccurate parameters and refine these parameters later. In the following, we distinguish between parameter determination before the first use of a system and during the use of a system.

6.4.1 Parameter Determination Before Use

We have identified four ways to acquire the required data for a decision-theoretic model before the first use of a system: (1) Appropriate data already exists, (2) necessary data can be gained with appropriate experiments, (3) it is possible to obtain judgments from experts, or (4) task analysis models can be used to determine the required data.

Existing Data The first example that we have mentioned in Chapter 1 involves information about train connections. Some of the information needed for a decision-theoretic model to plan a train trip is readily available, for example, in terms of a time table. The costs involved in the planning problem at hand correspond to the durations of getting from one station to another or waiting at a train station for a connection, which can both directly be read out from a time table. About other information required in this context, the railway company might already have collected statistics, for example, on train delays. Here, it will rather be a question of the railway company's willingness whether such information is provided or not, for example, if it makes the service of the railway company appear in a bad light.

Experiments We have shown with the experimental study in Section 3.1.1 how the parameters for a decision-theoretic model of the corresponding experimental setting can be determined exactly. This involves measuring the time that the system needs for its deterministic actions, averaging the time that a number of users need for a particular action, and computing probabilities for transitions from the relative frequencies of the outcome of a number of users' actions.

Expert Judgments In some cases, experts can provide information that can be incorporated in a decision-theoretic model. For example, if the railway company does not collect statistics about train delays, the software developer can ask conductors about the frequency of train delays on a particular connection. In this case, commuting passengers might be considered as experts equally well. In the case of providing assistance for operating a technical device, psychologists might be able to provide information about how well people memorize and recall sequences of instructions of a particular length. In a shopping scenario, a sales clerk might, for example, provide information such as how long people usually stand in line at the cash desk.

Where probabilities are involved, good estimations are sometimes sufficient to provide a basis for a decision-theoretic model that yields reasonable results. This is because sometimes the relative weight between probabilities is particularly important. For example, when estimating the probability that a particular user finds a book in one of two stores X and Y, it might be reasonable to estimate the probability that the user finds a book in store X as being twice as high as that he finds a book in store Y (e.g., because store X is twice as big as store Y). Similarly, one can guess that entering a 16 digit credit card number to a credit card phone is, for example, four times as error prone as entering a four digit expiration date. Experts may base such estimations on the results of experiments that study a situation *similar* to the one that is to be modeled.

Figure 6.3 illustrates a simple situation in which decreasing all probabilities by 50% results in the same policy if both achieving secondary goal A and secondary goal B are associated with a positive reward and the costs of all transitions are equal in the decision-theoretic model. For a user at location L_0, the policy, in particular the recommendation sequence to go to the goal via location L_1 and L_4, does not change if all probabilities are changed proportionally.

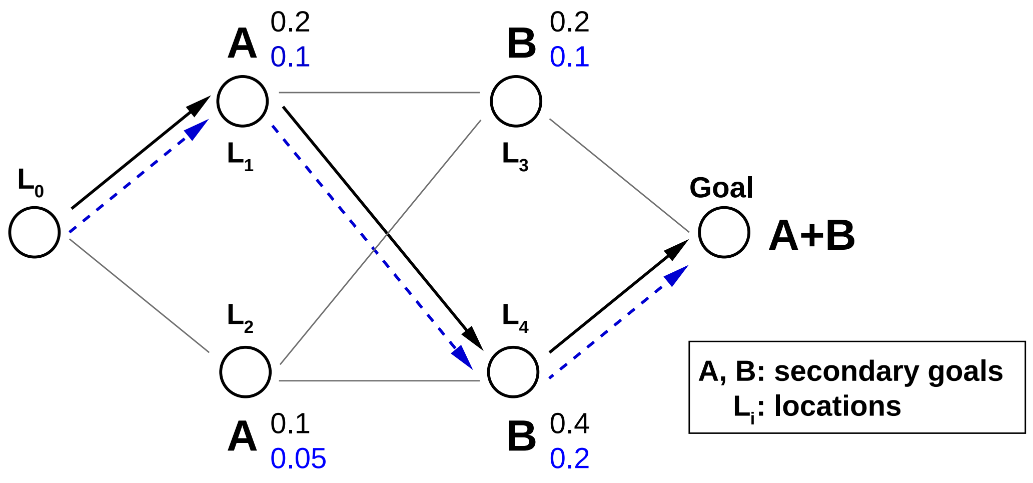


***Figure 6.3**: Unchanged policy for proportionally decreased probabilities.*
(Secondary goal A *can be achieved at location* L_1 *and* L_2*, secondary goal* B *can be achieved at location* L_3 *and* L_4*. The numbers represent the probabilities that a secondary goal* A *or* B *is achieved by going to the destination via a particular location. With unchanged uniform transition costs and unchanged rewards for achieving* A *and* B*, bisecting all probabilities in the decision-theoretic model does not change the policy—cf. the upper labels/solid arrows and the lower, underlined labels/dashed arrows—in this simple example.)*

After all, the system has to make a decision in favor of one out of a restricted number of options. The sensitivity analysis in Section 4.2.5 has shown that sometimes the exact probabilities are not very important. The system developer can always make a sensitivity analysis if he is worried about the inaccuracy of probabilities elicited by experts. If the numbers acquired before use allow the system to come up with some useful service for the user, then it might be sufficient to refine the parameters during the use of the system.

Nilim and El Ghaoui (2004) have recently proposed an algorithm to compute robust solutions of Markov decision processes, that is, solutions that are guaranteed to be robust with respect to estimation errors on the state transition probabilities within given boundaries. Bounded parameter Markov decision process have also been explored by Givan, Leach, and Dean (1997), who give references to earlier work on this subject in the operations research literature.

Task Analysis Techniques Task analysis techniques, such as the GOMS[5] family of user interface analysis techniques described by John and Kieras (1996), can sometimes be used to calculate information that is important for a decision-theoretic model. The analysis of descriptions of user procedures can provide quantitative predictions, such as the expected average execution time of an action performed by the user in the phone assistance example.

Similarly, for example, the expected walking time from one location to another, given the distance between the two locations and the average speed of a user, can be calculated for the navigation assistance example. Moreover, using distance measures instead of time measures to describe transition costs provides for a user-independent specification. To compute costs in terms of time, the average speed of a particular user might be provided by a user model and used to adjust the general model for the particular user before the computation of a policy.

6.4.2 Parameter Determination During Use

In the Chapter 7, we describe two user studies on the acceptance of a PDA-based decision-theoretic shopping guide (see Section 7.2 and 7.3). In the first user study, we were able to construct a reality that corresponded to our model. In the second user study, the estimation of the parameters was based on a thorough analysis of the shopping mall. Similarly, the estimation of the parameters can be accomplished for system for real use. If this initial specification is accurate enough to ensure that the system is of some use to shoppers, further data (e.g., times to get from A to B, success in finding a particular item in a particular shop) can be obtained from the PDAs of shoppers who have used the system, provided that they are willing to take a few seconds to allow data about their searches to be (anonymously) transmitted back to the central system. In addition to supporting estimates of the parameters, this method opens up the prospect of introducing some form of social navigation (see, e.g., Riedl, 2001 or Svensson, H¨o¨ok, Laaksolahti, & Waern, 2001) or collaborative filtering (see Goldberg, Nichols, Oki, & Terry, 1992; an overview is given by Resnick & Varian, 1997) in the shopping scenario.

6.5 Synopsis

The development workflow presented in this chapter describes how to get from a user-adaptive system planning problem to an appropriate decision-theoretic model and a good solution to the problem, respectively. First, we have described the workflow for scenarios of low complexity. For such scenarios, we have illustrated the overall chronology of the development process and the relations between a user-adaptive system problem specification and the basic decision-theoretic planning concepts. Then, we have then extended the workflow description such that it can be applied for scenarios of higher complexity, involving goal-priorized decision-theoretic planning. In this context, we have also discussed how a more general goal ranking with n-ary goals can be applied in domains even more complex than the example considered in Chapter 5. With regard

[5]GOMS stands for **G**oals, **O**perators, **M**ethods, and **S**election rules.

to the extended workflow description, the interleaving of macro-level and micro-level decision-theoretic planning was of particular interest. The last section of the chapter was dedicated to the determination of the quantitative parameters required for a decision-theoretic model. Thereby, we have distinguished between methods for the parameter determination before and during the use of a system.

The description of the development workflow may serve as a basis for a tool that supports the developer of a user-adaptive system when incorporating decision-theoretic planning to enhance a system's adaptation capabilities. The actual development of such a tool is beyond the scope of this thesis.

7 USABILITY ISSUES

In this chapter, we focus on two usability issues of systems that use decision-theoretic planning to give navigation recommendations. The first issue is the general user acceptance of a system giving navigation recommendations on the basis of decision-theoretic planning. We report two studies in which we studied if users accept a decision-theoretic shopping guide on a mobile computing platform. The first study was conducted in a mock-up of a shopping mall (Bohnenberger, Jameson, Kr¨uger, & Butz, 2002a, 2002b), which we established on two floors of the computer science building at Saarland University. For this study, infrared localization technology was used, such that the shopping guide under examination can be characterized as *location-aware*. The second study was conducted in a real shopping mall (Bohnenberger, Jacobs, & Jameson, 2005). Here, the assumed location awareness of the system was simulated by the experimenter, who imitated the signals that would otherwise have been transmitted by infrared beacons. Decision-theoretic planning was used to determine the system behavior in both user studies. The shopping guide used in the second study is an enhancement of the system used in the first study and takes the lessons learned in the first study into account.

One important result of the first user study was that people usually want to stay in control while they are guided by the system. Instead of being led blindfolded through a building, the users want to know why the system gives a particular recommendation. A system that makes its decisions based on a decision-theoretic planning approach has in the first instance not much information to offer to the user in order to justify or explain its decisions. It can always point out that the selected recommendation offers the highest expected utility for the user (according to the specified optimality criterion), but this will in general hardly be satisfying the user. Therefore, the second part of this chapter describes a user study (see also Bohnenberger et al., 2005) in which we analyzed what kind of information can serve as good explanation for recommendations based on decision-theoretic planning, and how such information should be represented. Finally, we describe techniques to aquire the necessary information to provide good explanations when they are requested.

7.1 User Acceptance of a Decision-Theoretic Shopping Guide

With user-adaptive systems, as in human endeavor more generally, intelligence alone does not guarantee success. It may be easy to equip an interactive system with knowledge and reasoning capabilities that are in principle highly relevant to the tasks that users perform with the system. It may be much more difficult to ensure that users will want and be able to benefit from this intelligence, given their customary patterns of activity and the constraints of their situations.

Part of the problem lies in the technical limitations of current wireless technology, which lead to long waits and frequent interruptions of connections, in addition to clumsy interaction with small devices (cf. Singh, Jain, & Singh, 1999). One remedy is to aim for designs that work well with the current limited technology, checking with users to see whether they really do work well enough.

Another type of problem concerns designs that do not adequately take into account the conditions under which mobile systems are used (e.g., while the user is in motion and/or is simultaneously engaging in some other activity). Only realistic tests conducted early enough in the design process can reliably prevent such problems (see, e.g., Sawhney & Schmandt, 2000 or Jameson, 2002).

In this section we report on an effort to apply this strategy in the design and development of an intelligent shopping guide based on AI techniques of decision-theoretic planning.

Combining Product Location With Navigation Support Location-aware mobile commerce systems that aim to bring together customers and products[1] have so far fallen into two main categories (see Duri, Cole, Munson, & Christensen, 2001, for more a extensive discussion and examples).

1. *Product location services*: A user $\mathcal{U}$ queries his system $\mathcal{S}$ about the availability of a particular type of product near his current location (see, e.g., the iGrocer system described by Shekar, Nair, & Helal, 2003, or the shopping guide described by Newcomb, Pashley, & Stasko, 2003).

2. *Location-dependent alerting services*: When $\mathcal{U}$ arrives at a given location, he is automatically notified about nearby products that are known to be of special interest to him (see, e.g., Shopper's Eye described by Fano, 1998, or the Shopping Jacket described by Randell & Muller, 2000).

In both of these schemes, the route of $\mathcal{U}$ is not planned ahead in any way, and the job of $\mathcal{S}$ is to help $\mathcal{U}$ do as well as possible in or near the given location[2]. Consequently, $\mathcal{U}$ may discover that no especially desirable products are available near his current location even though there exist more attractive ones at some location that he might just as well have visited.

The alternative approach to be examined in the first two user studies described in this chapter take into account the fact that people often have considerable freedom in determining what particular locations to visit. To take an example that involves a small geographic region, consider a user who wants to walk through a large shopping mall on the way to some other destination and would like to pick up a few items along the way—providing that he can do so without investing too much extra time. Instead of taking the shortest route through the shopping mall and finding out about products that happen to be available along that route, he may well be willing to take a longer route that leads him by places where he is more likely to find the desired products. Suppose now that $\mathcal{U}$ is to be guided through the shopping mall by a system running on a handheld device:

[1]To facilitate exposition, we will use the term *products* even when what is really meant is products, services, and combinations of both. As in preceding chapters, the symbol $\mathcal{S}$ will denote the system under discussion, while $\mathcal{U}$ will refer to a user of that system.

[2]A system may provide navigation support to guide $\mathcal{U}$ from his current location to the exact nearby place where he can obtain the recommended product; but the current location is still treated as given.

It makes sense for the system to apply techniques similar to those used in the more sophisticated route planning systems for automobiles, which choose a route according to a mixture of criteria such as driving time, gas consumption, and scenic attractiveness.

On the other hand, the planning problem in the shopping mall has an important difference from the typical route planning problem: the need to deal with the uncertainties that are inherent in almost any attempt to match customers with products. For example, if $\mathcal{U}$ is looking for a book, but the exact type of book is rather indefinite (e.g., "Something amusing to read on the plane"), there will be uncertainty because: Even if $\mathcal{S}$ has exact information on the available books, $\mathcal{S}$ cannot predict with certainty if $\mathcal{U}$ will find one of them that fulfills his requirements.

As is explained in the preceding chapters, $\mathcal{S}$ can deal adequately with this uncertainty if it computes not a fixed plan but a policy in the sense of decision-theoretic planning. A policy can specify, for example, that if $\mathcal{U}$ doesn't find a suitable book in Bookstore *A* he should make a slight detour to try the nearby Bookstore *B*.

One of the issues to be addressed in this chapter is whether this type of decision-theoretic planning can actually lead to more effective shopping in the context of a working system. And even if the shopping is more effective, how do users evaluate it subjectively? We have first addressed this question within a mock-up of a shopping mall, have then improved the system on the basis of what we have learned in the mock-up case, and have finally tested the improved system in a real shopping mall (yet, without the infrared localization technology). The basic method is applicable on very different scales, such as the shopping within a single large store or an entire city.

7.2 Study in a Mock-Up of a Shopping Mall

We have implemented and tested a PDA-based system that gives a shopper directions through a shopping mall on the basis of (a) the types of products that the shopper has expressed an interest in, (b) the shopper's current location, and (c) the purchases that the shopper has made so far. The system uses decision-theoretic planning to compute a policy that optimizes the expected utility of a shopper's walk through the shopping mall, taking into account uncertainty about (a) whether the shopper will actually find a suitable product in a given location and (b) the time required for each purchase (cf. Figure 7.1). To assess the acceptability of this approach to potential users, we constructed a mock-up of a shopping mall with 15 stores on two floors of the computer science building at Saarland University. Each of 20 subjects in our study shopped twice in the mall, once using our system and once using a paper map. The subjects completed their tasks significantly more effectively using the PDA-based shopping guide, and they showed a clear preference for it. They also yielded numerous specific ideas about the conditions under which the guide will be useful and about ways of increasing its usability.

7.2.1 Realization of the Shopping Guide

For concreteness, we will explain the workings of the shopping guide with reference to the hardware used and the (fictitious) shopping mall that we set up for the user study. Consider a user $\mathcal{U}$ who enters a mall with the intention of buying five products: a CD, some bread, a newspaper, a PDA, and a plant. As the partial map in Figure 7.2 indicates, products of each of these types are available in one or more stores. But when deciding which stores to visit, $\mathcal{U}$ needs to take into account the facts that (a) it is not certain that he will find a truly suitable product even in a potentially relevant store; and (b) stores differ in the time it takes to wait in line to pay for a product.

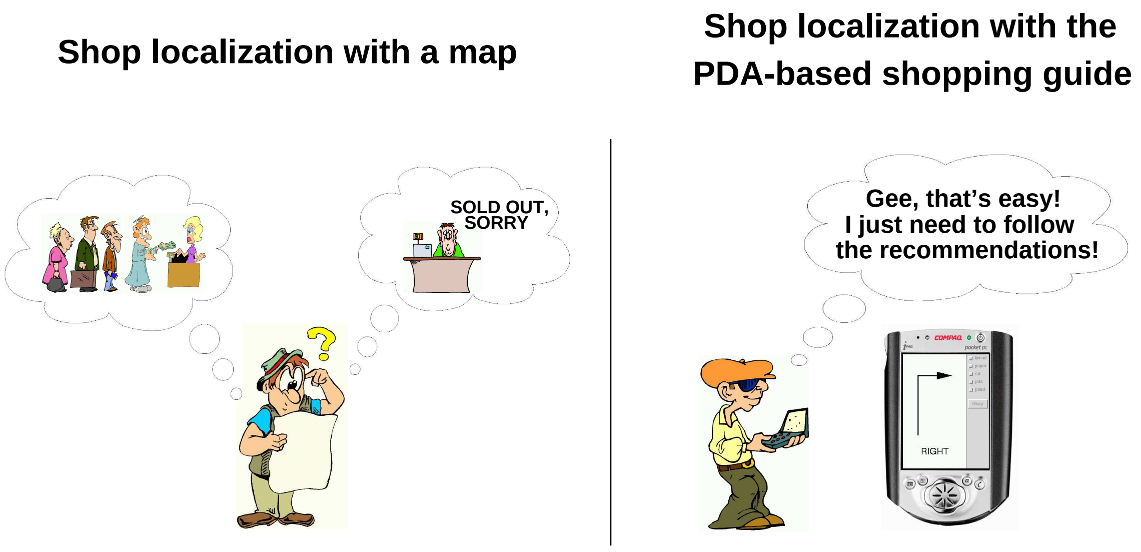

Shop localization with a map
Shop localization with the PDA-based shopping guide
SOLD OUT, SORRY
Gee, that's easy! I just need to follow the recommendations!
COMPAQ
pocket pc
RIGHT

***Figure 7.1**: Navigating in a shopping mall.*
(Using a conventional paper map, the planning of a shopping expedition can be a complex task. The navigation during a shopping expedition can be facilitated with a PDA-based shopping guide. In particular, uncertainty about product availability and expected times to stand in line can be taken into account using a decision-theoretic planning approach.)

The goal is to allow $\mathcal{U}$'s PDA to determine at each point in time which store $\mathcal{U}$ should visit next. Since the necessary planning (described below) is computationally expensive, it is performed on a stationary computer at which $\mathcal{U}$ enters a specification of the desired products through some suitable form of interaction (e.g., menu selection on a touch screen). The resulting policy is compact enough to be transferred to $\mathcal{U}$'s PDA, and it contains all the information that the PDA-based application requires.

As Figure 7.3 indicates, the PDA guide should at any moment direct $\mathcal{U}$ towards the next store by displaying a navigation instruction on the PDA screen; $\mathcal{S}$ will determine the current location and orientation[3] by receiving an infrared signal from one of a number of beacons affixed to the walls, and $\mathcal{S}$ will know what products $\mathcal{U}$ has already found because $\mathcal{U}$ will check each one off after purchasing it.

7.2.1.1 Hardware

The particular hardware configuration employed in the user study (see Figure 7.3) gives an idea of the typical hardware requirements: The specific PDA was a Compaq iPAQ 3620 Handheld, which can receive signals from the small *infrared beacons* via its built-in *IrDA interface*. We mounted 30 of these beacons at decision points (i.e. turns, crossings, stairs) in the building in which the study was conducted, each at a height of about 2.5 meters. The beacons emit a 16 bit ID twice a

[3]A beacon transmits signals only within a small angle: therefore receiving signals from a beacon means pointing with the PDA's infrared interface into the opposite direction the beacon transmits to. This information can be used by $\mathcal{S}$ to present orientation-aware recommendations.

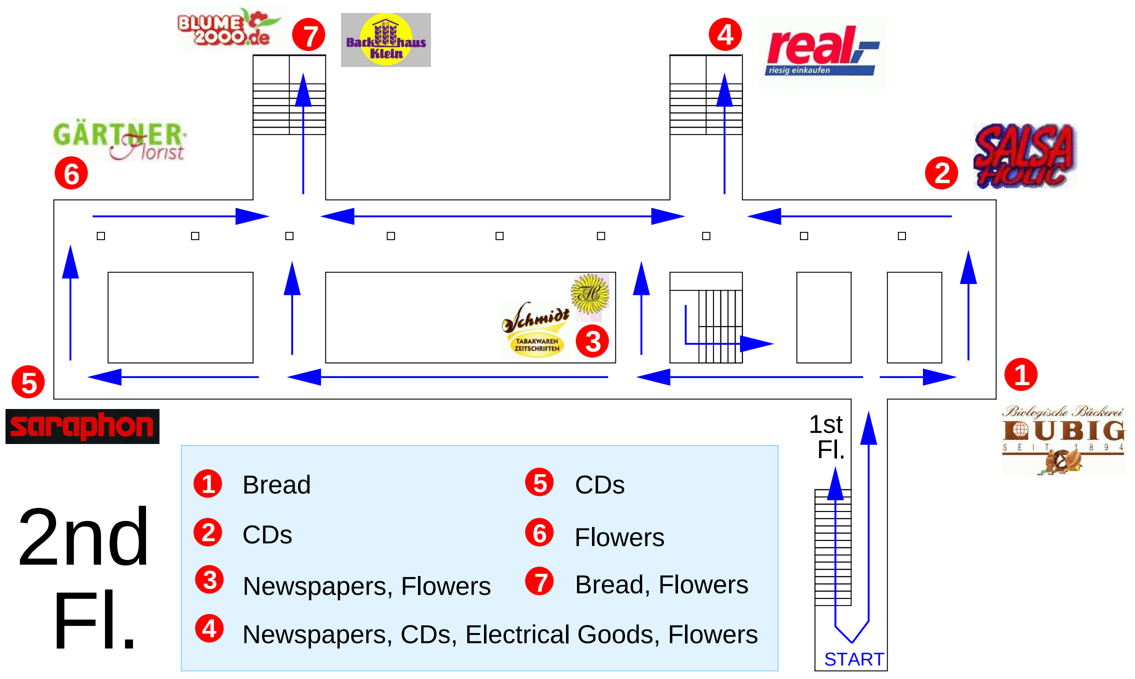


***Figure 7.2**: Part of one of the maps representing the shopping malls for the first user study. (Icons represent stores, arrows indicate directions the subject is allowed to walk to. The arrows can be interpreted as some sort of one-way passages, which guarantee that the subject will reach the mall exit without passing along any of the shops more than once.)*

second, and they have a transmission range of about 7 meters. Running on batteries, they do not need access to the power supply system or to a data network.

The PDA software for the shopping guide is implemented in TCL/TK. When a beacon signal is received, the browser displays the corresponding navigation recommendation according to the precomputed navigation policy, taking into account both the beacon's location and the products that $\mathcal{U}$ has checked off on the shopping list shown on the right-hand part of the screen. If a beacon ID corresponding to a store location is received, the items offered in that store are highlighted in the shopping list.

7.2.1.2 Decision-Theoretic Model

The model used for the decision-theoretic planning is similar to the one described in Chapter 4.2.3. Our basic assumptions are: (a) $\mathcal{U}$ moves from a starting point to a destination (entry and exit of the shopping mall) via distinct positions in the building, (b) on his way he wants to find as many items of his shopping list as possible in a relatively short time, (c) in each of the stores he has a chance to purchase some of the items needed, and (d) in each of the stores in which he finds an item, he will spend a certain amount of time waiting in line to pay for it.

We encode this information in the state features and the transitions between the states as follows: A state is defined by (a) the position of an infrared transmitter, (b) a list of the needed shopping items (products) and the items already purchased, and (c) a flag indicating the status

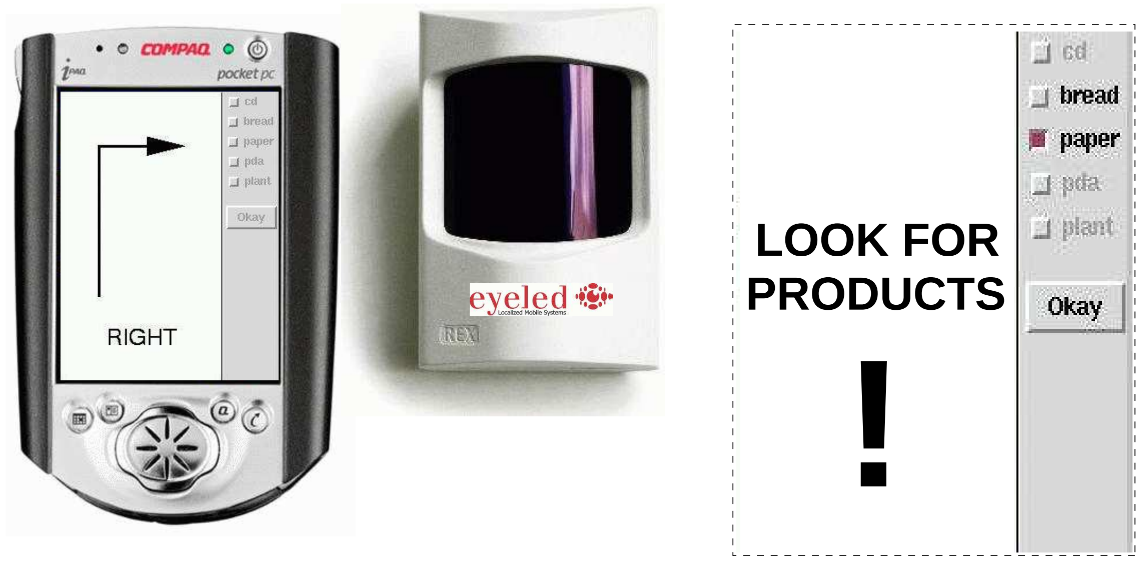


***Figure 7.3**: The hardware of the PDA-based shopping guide.*
(Left: Example display on the PDA, which receives location signals from a beacon (middle). Right: Close-up of part of the display in the situation where the user has arrived at a relevant store. $\mathcal{U}$ checks off the items found in the current store and clicks "okay" to proceed with the next navigation recommendation.)

of $\mathcal{U}$'s shopping procedure at that position—a feature that is significant only if there is a store at the position in question. This flag has one of four values: (1) NEUTRAL, indicating that $\mathcal{U}$ has not yet checked if any of the desired products is available at the current store (2) NOT FOUND, indicating that none of the products needed is available in the store, (3) FOUND, indicating that $\mathcal{U}$ has determined that at least one of the products needed is available; and (4) BOUGHT, indicating that $\mathcal{U}$ has bought at least one product.

Figure 7.4 shows an example of the system dynamics. Suppose $\mathcal{U}$ approaches a junction located at position 2. Whether or not he has already purchased product A (corresponding to the two states shown for position 2), $\mathcal{S}$ has the choice between recommending LEFT or RIGHT, which would lead $\mathcal{U}$ to position 6 or position 3, respectively. Each of these transitions has a cost that depends on the time it will take $\mathcal{U}$ to reach the next position.

Suppose that $\mathcal{U}$, who has not yet purchased product A, approaches the store at position 3, which offers product A. In this case, $\mathcal{S}$ can recommend RIGHT to have $\mathcal{U}$ pass by the store and go ahead towards position 4. But $\mathcal{S}$ can also recommend that $\mathcal{U}$ enter the store and LOOK FOR PRODUCT A. In this case, $\mathcal{S}$ needs to consider what $\mathcal{U}$'s chances are of finding product A in this store. Moreover, looking for product A is associated with a cost: the usual time needed to check if product A is available or not. If $\mathcal{U}$ finds product A in the store, it remains uncertain how long he will have to wait in line to pay for it. This waiting is modeled in a simple fashion by a circular WAIT-transition which is associated with a particular probability of the form $(n-1)/n$ for an integer n. This model would be entirely accurate if (a) the cashier required a constant amount of time to deal with each customer, (b) there were always n customers waiting, and (c) the cashier always chose the next customer at random from the n who were waiting. These assumptions will seldom be satisfied exactly, but they do make it possible to approximate a broad range of

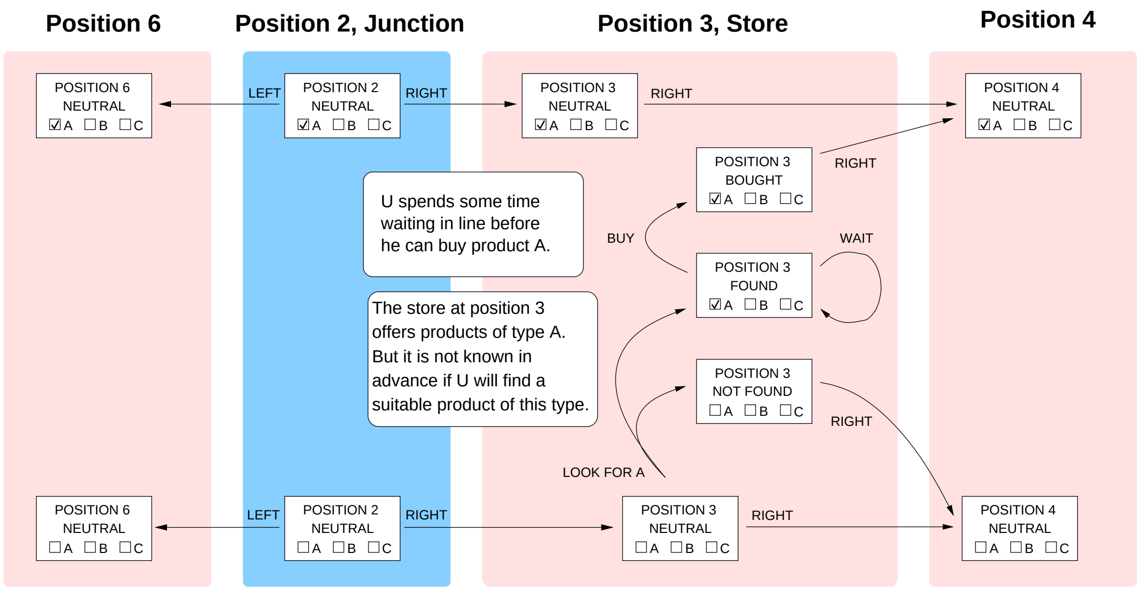

Position 6
Position 2, Junction
Position 3, Store
Position 4
POSITION 6 NEUTRAL ☑A ☐B ☐C
LEFT
POSITION 2 NEUTRAL ☑A ☐B ☐C
RIGHT
POSITION 3 NEUTRAL ☑A ☐B ☐C
RIGHT
POSITION 4 NEUTRAL ☑A ☐B ☐C
POSITION 3 BOUGHT ☑A ☐B ☐C
RIGHT
U spends some time waiting in line before he can buy product A.
BUY
WAIT
POSITION 3 FOUND ☑A ☐B ☐C
The store at position 3 offers products of type A. But it is not known in advance if U will find a suitable product of this type.
POSITION 3 NOT FOUND ☐A ☐B ☐C
RIGHT
LOOK FOR A
POSITION 6 NEUTRAL ☐A ☐B ☐C
LEFT
POSITION 2 NEUTRAL ☐A ☐B ☐C
RIGHT
POSITION 3 NEUTRAL ☐A ☐B ☐C
RIGHT
POSITION 4 NEUTRAL ☐A ☐B ☐C

***Figure 7.4**: Illustration of the modeling underlying the decision-theoretic planning employed in the shopping guide.*
(Explanation in text.)

more realistic waiting processes. Once $\mathcal{U}$ has finally BOUGHT product A, $\mathcal{S}$ will give the next navigation recommendation: in this case the only option is RIGHT.

The Markov decision process described here is *fully observable*: We can assume that the system always knows with certainty $\mathcal{U}$'s position, the items needed and the items already purchased. The rewards for the goal states, which are associated with the position at the mall exit, are a function of the number of products that $\mathcal{U}$ has bought during his shopping expedition.

Given this representation of the planning problem, we can use value iteration (see Section 3.1.4) to compute the navigation policy. This takes no longer than some seconds on a workstation since the number of state features is small in the problem at hand. A part of the policy that was applied in this user study is listed in Appendix G. Its syntax is in the style of the tcl/tk programming language, such that it can directly be applied by a tcl/tk interpreter, which we have used to visualize the navigation recommendations on the handheld device for the user study.

The modeling underlying the decision-theoretic planning can in principle take into account a variety of factors other the ones included in our current implementation, such as, for example, the difficulty of correctly following each instruction (cf. Section 4.2.3).

7.2.2 First User Study: Method

The overall purpose of the first user study was to check the objective effectiveness and subjective acceptability of the shopping guide just described, with regard to both its planning methods and its hardware solution.

In order to have a point of comparison for the results with the PDA guide, we also had the subjects perform the same basic task with a conventional paper map (see the example in Figure 7.2). The map was designed to be at least as helpful as the best maps that can be found in shopping malls: It was portable (in contrast to the frequently found wall-mounted maps, which burden users' memory), and it showed all of the relevant information that could reasonably be expected to be found on a paper map. This information did not include probability distributions for waiting times or for the likelihood of finding suitable items of particular categories.

Mock-Up of the Shopping Malls One configuration of 30 beacons was installed on two floors of the main Computer Science building at Saarland University. Two alternative shopping malls (equivalent in navigational complexity), each comprising 15 stores, were defined, so that each subject would be able to perform a distinct shopping task with each of the two types of guidance. Figure 7.2 shows the upper floor of one of these malls. It was not feasible to decorate the building with physical representations of stores, but the subject could always see from the map or the PDA display when he had reached a given store.

In addition to the physical constraints on movement enforced by the architecture of the building, we added some further constraints that might be found in a more complex setting: Some segments of hallways were designated as "one-way passages", as is indicated by the arrows in Figure 7.2. Subjects using the map were instructed not to walk in the wrong direction on these segments, and the instructions given by the PDA guide likewise obeyed these constraints. Analogous constraints are found in real shopping malls, for example, in the form of escalators that run in only one direction or entrances through which shoppers are not allowed to exit (and vice-versa).

Since we designated a relatively large number of one-way segments in our malls (partly for technical reasons that will be discussed below), our malls may have been untypically challenging in this respect. Because of the one-way segments, in many situations a subject found that there was no permissible path by which he could reach a relevant store that offered a product that he was (still) looking for.

Experiments on consumer behavior in which subjects are only pretending to buy are routinely criticized. Yet, unlike in most e-commerce studies, not the buying of products itself, but rather the navigation to find the products is the central aspect of the first study described here. A drawback we are aware of is that the subjects were the only shoppers in the mall at a time, that is, they were able to do their shopping completely undisturbed by other visitors that would usually be present in a real shopping mall. To come up against this criticism, we have conducted the second user study—reported in Section 7.3—in a real shopping mall.

Shopping Tasks Each subject employed the paper map and the PDA guide in two successive trials.[4] The subjects' task in each case was to buy as many products as possible from a shopping list with 5 categories: bread, a plant, a CD, a newspaper, and a PDA. Each of the following combinations and temporal orderings was employed with 5 of the 20 subjects:

- PDA in mall 1; map in mall 2
- PDA in mall 2; map in mall 1
- map in mall 1; PDA in mall 2
- map in mall 2; PDA in mall 1

[4]A within-subject design was chosen over a between-subject design because of (a) our expectation that individual differences would be large; and (b) our desire to hear the comparative comments of subjects who had used both types of shopping guidance.

Therefore, neither learning effects nor differences between the two malls could bias the overall pattern of results.

Subjects We recruited 20 subjects (15 of them female). Of these subjects, 15 were students, 3 were employed persons, 1 was a homemaker and 1 was an unemployed person. Only 2 of the subjects had used a PDA before, neither of them for a navigation task. Most of the subjects knew what a PDA is and what it is normally used for. The subjects' reward was made up of (a) a fixed amount of money (15.00 Deutsche Mark[5]) plus (b) an additional performance-dependent amount: They received 2.00 Deutsche Mark for each product they managed to "buy", but 0.01 Deutsche Mark was deducted for each second that they spent shopping. This cost of time was explained by analogy to the parking costs incurred during shopping.

Procedure Each subject performed the tasks individually under the guidance of an experimenter. First, the experimenter explained the nature of the tasks. He then familiarized the subject with the paper map (e.g., how the paths between stores were visualized) and with the usage of the PDA guide (e.g., how to hold the device to ensure good reception of the beacons' signals). During the explanation of the reward policy, the subject was instructed not to race through the building in order to maximize his financial reward but rather to walk at a normal pace, obeying the constraints shown on the map (when using the map) or following the PDA guide's directions.

In each of the two trials, the subject began at the mall entrance and was followed around by the experimenter[6]. When the subject reached a store, the experimenter took on the role of the sales clerk at that store: He informed the subject about the availability of the desired product(s); and if at least one was available, he determined how long the subject would have to wait in line to buy it, enforcing the waiting time with a stopwatch.

To ensure that the results concerning product availability and waiting time were consistent with the PDA guide's model of the shopping mall, the experimenter made these determinations with the help of a small program on his own PDA that generated random numbers and compared them with the probabilities specified in the model of the shopping mall.

After both trials had been completed, the subject filled in a questionnaire and was interviewed briefly.

7.2.3 First User Study: Results

The study proceeded without problems; the subjects showed considerable interest in the problem and the two guidance methods presented, and they supplied detailed comments on the questionnaire and in oral discussion.

7.2.3.1 Objective Results

As illustrated in Figure 7.5 (left), the PDA guide did enable subjects to complete their shopping task 11% faster on the average than with the map, the difference being statistically significant according to a paired t-test ($t(19) = 2.65, p < .05$). This difference in itself may not seem to constitute a practically important advantage, but it seems more noteworthy when we consider two factors:

[5]2.00 Deutsche Mark roughly correspond to 1.00 Euro.

[6]The procedure of the user study is illustrated in a short video that can be requested by the author.

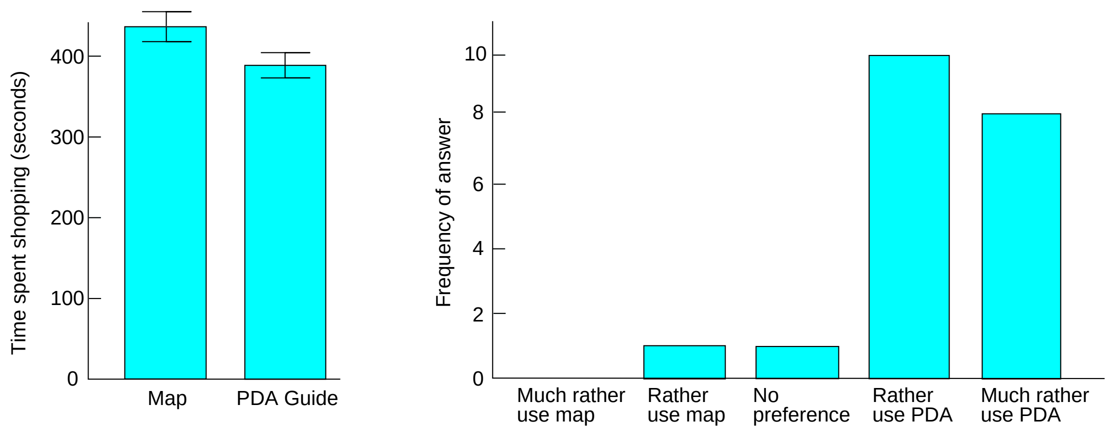


***Figure 7.5**: Results of the first user study.*
(Left: Mean values for the quantitative performance index for each of the two types of shopping guidance. Each bracket indicates the standard error of the corresponding mean. Right: Frequency distribution of answers to the question "Which method would you rather use if you had to perform more shopping tasks like the ones you performed in this study?")

- In the map condition, subjects were allowed to study the map as long as they wanted before starting to shop, and this study time was not counted. In the PDA condition there was nothing to study, and the subjects set off immediately.
- Whereas all subjects were familiar with paper maps, none had previously used a PDA navigation aid like the one in this study, and only 2 had used PDAs at all. So the time for the PDA condition includes any time that subjects might have required to get used to using the PDA guide and to dealing with the infrared signals from the beacons.

The map and PDA conditions were just about identical in terms of the number of products that subjects found; the very slight advantage of the PDA condition does not approach statistical significance. It is not surprising that there is no reliable difference here: The number of products found is a relatively coarse-grained variable, the possible values being just the integers between 0 and 5; and the situation was set up in such a way that in general most of the desired products could be found, so there was little room for large differences. This circumstance is also known as *ceiling effect*.

It is unclear how well these quantitative results would generalize to other concrete scenarios and to qualitatively different users. Still, the results for this study demonstrate that the PDA guide can fulfill its promise of enabling shoppers to complete their task more effectively despite the two handicaps just listed.

7.2.3.2 Subjective Results

While the objective effectiveness of the PDA guide may be a necessary condition for its ultimate acceptance, it is not a sufficient condition. An important question is how attractive subjects would find the prospect of using such a guide in real shopping (bearing in mind the fact that they had no opportunity to experience it in truly realistic conditions). As an assessment of overall preference, the questionnaire included a question that asked which method they would prefer to use if they had to perform further shopping tasks like the ones they had just completed. Figure 7.5 (right) shows that the subjective preference is much clearer than the objective advantage. We cannot rule out that some subjects were biased in favor of new technology or such that they wanted to please the experimenter (cf. Landauer, 1997 for a discussion of the bias problem).

As another way of assessing preference for the PDA guide, the questionnaire asked how much time subjects would be willing to spend initializing the PDA guide for each new shopping expedition. All but 6 subjects wanted to keep preparation time below 2 minutes.

It might be suspected that this difference (and perhaps the objective speed difference as well) were due to specific inadequacies in the design of the paper map. But subjects' responses to the relevant questions made it clear that they saw fewer correctable problems with the map than with the PDA. Nevertheless, the usage of multiple maps put up in important locations of the building (and designed strictly according to principles as they are stated by, e.g., Darken & Peterson, 2001) instead of a map carried along by the subjects might facilitate the subjects' orientation in the map condition.

We had tried to ensure a fair comparison by designing each of the two "interfaces" as appropriately as possible. As a check for the success of this effort, the questionnaire included two questions which asked to what extent the subject thought that the map (or the PDA interface) could have been designed so as to support their tasks better. The distributions of the responses were almost identical, with more than half of the subjects judging in each case that the design "couldn't have been better".

In sum, although subjects did have some important suggestions for the improvement of both guidance methods (to be discussed next), they showed a clear preference for the type of experience created by the PDA guide.

In order to understand these quantitative subjective results and to judge how they can be generalized, we need to understand the experience and reasoning that underlie them. On the questionnaire, subjects were encouraged to add verbal comments to their ratings. In addition, the questionnaire contained open questions asking about (a) problems and sources of irritation with the map and the PDA guide; and (b) ways of improving the PDA guide. Since in many cases closely related comments were made to different questions, we summarize them here not in terms of the individual questions but in terms of the overall pattern of ideas that emerged.

7.2.3.3 Reasons Given for Preferring the PDA Guide

The reasons that the subjects have given for preferring the PDA-based decision-theoretic shopping guide over the paper map can roughly be summarized in three categories: (1) The subjects felt that they had to spend less time and effort compared to using the paper map; this subjective assessment was approved by the quantitative results reported in Section 7.2.3.1; (2) they rated the cognitive effort for using the PDA guide less than for using the paper map; and (3) they pointed out the reduced need for thinking about orientation when using the PDA guide.

Saving Time and Effort The most obvious advantage of the PDA guide is that the shopper can typically complete his tasks in less time than with a map. As one subject mentioned, there is also a corresponding savings of effort in moving around the shopping mall—an advantage which some users may find appealing even if they are not concerned about time, for example, if their mobility is for some reason limited or if the mall is much larger than the one used in this study. These comments confirm that subjects recognized the superior ability of the PDA guide to deal with complexities such as uncertainties about product availability and waiting time—despite their unfamiliarity with systems of this sort and with the underlying technology. The comments also confirm that the potential advantages conferred by the PDA guide's knowledge and reasoning were not seen by subjects as being outweighed by problems of usability or unfamiliarity. Altogether, 12 subjects gave comments that corroborate this result.

Less Cognitive Effort and Frustration A less objectively measurable advantage is the reduced need for mental exertion with the PDA guide. As one subject put it, "You don't have to plan where you're going to go, or think for yourself." When using the map, most subjects recognized the advisability of planning ahead, though it would not have been feasible for them even to attempt the type of decision-theoretic planning that is applied on the PDA guide. A typical strategy was to

- plan a few steps ahead, assuming that the products they wanted to get in the nearby stores would be available; and
- replan whenever an product proved unavailable.

In addition to leading in general to suboptimal solutions, even this simple strategy requires more mental exertion than simply following the advice given by the PDA guide. Moreover, users found it frustrating to have to abandon part of a plan that they had already spent some effort devising. Altogether, 6 subjects gave comments that point into this direction.

Less Need for Thinking About Orientation A small number of subjects pointed out that with the map, they continually had to ensure that the map was oriented correctly and to keep in mind where they were. By contrast, the PDA guide simply gave directions relative to their current position and orientation; it can take into account the direction the user is facing to a certain extent because each beacon only transmits signals within a limited angle.[7] Our subjects' problems when navigating with the map are similar to those already unveiled in a more general analysis by (Thorndyke & Hayes-Roth, 1982) about problems that people may have with orientation under different conditions. Altogether, 5 subjects either admitted that they have problems with maps in general or pointed out explicitly that they appreciate not having to think about orientation when using the PDA guide.

7.2.3.4 Perceived Drawbacks of the Current PDA Guide

Even subjects who were on the whole enthusiastic about the PDA guide mentioned some limitations, most of which we were able to eliminate through improvements in its design. For example, the system used in the study reported in Section 7.3 exhibits new features that help to prevent the user from not having the big picture or feeling like not being in control. The other drawback that some subjects have mentioned—the need to attend to the PDA and to the beacons—is merely a question of technical capability of the infrared-based localization method.

[7]Where uncertainty about orientation was possible, the PDA display included an identifiable landmark, such as a flight of stairs, in the immediate vicinity of the user's current position.

Feeling of Not Having the Big Picture Although some subjects appreciated not having to think about their orientation and location, for other subjects this consequence was associated with a drawback: They felt almost as if they were being led blindfolded through the shopping mall—a feeling that was evidently not entirely satisfactory to them even if they were convinced that the system was leading them in an optimal way. Altogether, 9 subjects pointed out this drawback.

Proposals made by subjects for improving their overall feeling of orientation included some ideas that are familiar from GPS-based navigation systems:

- Display overview maps of (parts of) the mall. For example, while $\mathcal{U}$ is walking from one position to the next, an overview map showing his position within (some part of) the shopping mall could be displayed.

- Make it possible for $\mathcal{U}$ to be given longer sequences of directions. Instead of always being led from one beacon to the next, subjects sometimes prefer longer sequences of instructions (e.g., the graphical equivalent of "Go straight ahead, then take the next right and then the next left"). In addition to giving users a greater sense of knowing where they are going, such sequences can save time, because they reduce the number of occasions on which $\mathcal{U}$ needs to acquire information from a beacon and consult the PDA. On the other hand, longer sequences place heavier demands on $\mathcal{U}$'s working memory, and they lead to a greater likelihood that some instruction will be executed incorrectly. This type of tradeoff was investigated in the experimental setting described in Section 3.1.1. Chapter 3 showed that finding an optimal instruction sequence length is a task that can be handled within the framework of decision-theoretic planning.

- Give advance announcements of upcoming stores. Some subjects wanted to know not only what direction they should proceed in but also what they could expect to find there. For example, if the PDA displayed along with the arrow a message like "Saraphon (CDs), 40 m", $\mathcal{U}$ could utilize the walking time to prepare mentally for the task of looking for a suitable CD.

In addition to the advantages already mentioned, users who have more than the minimally necessary amount of information about their surroundings are in general better able to cope with problems with the technology, which will be discussed next.

Need to Attend to the PDA and to the Beacons The current realization of the PDA guide requires users to attend to a certain extent to the handheld device and/or to the beacons even while walking: $\mathcal{U}$ needs to know when he has reached the range of the next beacon, so that he can attend to the next instruction. $\mathcal{U}$ may therefore

1. look out for the physical beacons themselves,

2. aim the PDA explicitly at a beacon and/or,

3. keep looking at the PDA to see when a new instruction appears.

Altogether, 12 subjects reported difficulties with this procedure or described it as being inconvenient. The first two activities should be facilitated by more powerful beacons, so that their signals reach the PDA whenever the user comes into the relevant area. Stronger beacons (with a range longer than twice the range of the beacons used in the study) have already been developed by Eyeled GmbH shortly after the user study.[8]

The third activity can interfere with $\mathcal{U}$'s walking, for example, increasing the danger of colliding with other persons or stumbling on stairs. An obvious solution, mentioned by some subjects, is the use of acoustic stimuli, which could be delivered via a loudspeaker or a headset. The acoustic output can range from (a) acoustic signals that alert $\mathcal{U}$ to a change in the display to (b) the use of speech for all navigation instructions. We have incorporated this idea into the improved system that we used in the study reported in Section 7.3.

7.2.4 Further Issues

The aspects of the shopping guide that could be tested in the present study passed their tests fairly well, and the improvements that proved desirable can be realized without much difficulty. But the goal of deploying the system in real shopping situations raises some issues that were not (fully) addressed in the present study.

7.2.4.1 Increasing the Volume of Products Purchased

One important potential advantage of the PDA guide was *not* demonstrated in the results of the present study: the advantage that shoppers might find more products that they want to buy. But this advantage might well be achieved in situations with somewhat different properties than the situation of the present user study:

First, if the time available is more severely limited (e.g., because stores are about to close), shoppers with the slower map method will have to stop shopping before they have exploited all possibilities. In this case, shopping times will tend to be equalized, with the difference appearing in the number of products found.

Second, shoppers often have a choice between shopping or not shopping (e.g., on the way to their gate in an airport) or between shopping in mall *A* or shopping in mall *B*. If a given shopping mall, by providing a shopping guide, offers the prospect of getting the job done more efficiently and with less cognitive effort, this mall has an advantage over competing malls in attracting customers to shop there in the first place.

7.2.4.2 Applicability to Spontaneous Shopping Behavior

Especially recreational shopping is often highly spontaneous; a shopper may suddenly abandon any plans he may originally have had in order to pursue some attractive new option. However, in situations where it is very hard to predict what the shopper might do, it is hard to specify an accurate model as a basis for decision-theoretic planning. The approach presented here is therefore most applicable when the shopper has particular goals that he wishes to achieve within a limited amount of time.

Our approach does not, however, presuppose that the shopper is blindly tracking down specific products on a shopping list with maximal efficiency. If $\mathcal{U}$ is allowed to specify vague product descriptions like "something to read" or "a present for my 3-year-old daughter", $\mathcal{S}$ will lead $\mathcal{U}$ to

[8]This technical solution does not overcome the problem if $\mathcal{U}$ points with the interface to the fbor. Other technologies, e.g., bluetooth, might be employed to indicate the presence of a transmitter without any special requirements for holding the PDA.

relatively promising stores, and $\mathcal{U}$ can be as creative as he likes in deciding what (if anything) to buy there[9].

It is possible to extend the model that underlies the decision-theoretic planning to account for possible events such as $\mathcal{U}$'s spontaneously stopping to spend time at stores that he passes. The proportion of shoppers whose behavior is adequately fit by the model's assumptions will depend on the sophistication and the accuracy of the modeling effort.

7.2.4.3 The Role of One-Way Passages in the Shopping Domain

As we have already explained in Section 3.1.7, decision-theoretic planning based on Markov decision processes has an inherently high computational complexity. For the user study, we made some assumptions about the reality being modeled which ensured that decision-theoretic planning would be computationally tractable. In particular, the introduction of *one-way passages* made it impossible for subjects to visit any store twice, thereby making it unnecessary for the decision-theoretic model to include information about the stores that $\mathcal{U}$ has already visited. The decision-theoretic modeling described by Plutowski (2003), which we have already mentioned in the introduction of Chapter 5, does without the assumption of one-way passages, but consequently works on shopping domains with less locations than those that we consider.

7.2.4.4 Learning of Parameters

For the modeling, a great many specific quantitative parameters need to be specified, concerning matters like walking and waiting times and the probability of finding a given type of product in a given store. We have already discussed solutions to this problem in Section 6.4. In our study, we were able to construct a reality that corresponded to our model. For real use, one approach would be to estimate the parameters initially on the basis of a more or less thorough analysis of the shopping mall. If this initial specification is accurate enough to ensure that the system is of some use to shoppers, further data can be obtained from the PDAs of shoppers who have used the system.

7.2.5 Conclusions

The first user study has provided an initial demonstration of the feasibility and acceptability to users of several aspects of a new approach to intelligent location-aware shopping guidance. The method is characterized by (1) an integration of navigation assistance with product location; (2) the exploitation of decision-theoretic planning techniques that perform computations too complex for humans to approximate mentally; and (3) the use of robust location-awareness technology which is already available at reasonable cost.

Our study with 20 users in a semi-realistic situation yielded a number of insights into the reasons why users find the system acceptable—and the conditions on which this acceptability depends (e.g., unfamiliarity with the environment or time pressure). It also turned up some problems and limitations, along with ideas for overcoming them. The results of the first user study encouraged us to test an improved version of the shopping guide in a real shopping mall. The ensuing second user study is reported in the next section.

[9]Compare also to the shopping task of the second user study described in Section 7.3.2.

7.3 Study in a Real Shopping Mall

The first user study conducted in a mock-up of a shopping mall has revealed potential improvements of the PDA-based shopping guide. The flaws most often mentioned in the first study have been eliminated in an improved version of the system. To help the user keep the big picture of what he is doing (or what the system recommends him to do, respectively) the new shopping guide features overview maps including the current location of the user and the location of the next recommended store. Photographs of stores are displayed on the PDA screen once the next goal is reached. This helps the user recognize which stores exactly he is supposed to visit next. Generally, the visual output of the new system is more attractive and lively, yet, free from unnecessary details that might disturb the user. Also the audio output has been improved.

As the first study was routinely criticized for not being conducted in a realistic environment, we took the opportunity to test the improved system in a real shopping mall. This way, the subject uses the shopping guide in an environment where he is surrounded by other people, is possibly disturbed by other people, and should take care about not disturbing other people by bumping into them when concentrating too much on the interaction with the system. We wanted to find out if this circumstance restricts the usage of a PDA-based shopping guide, and if it has a negative influence on the results of the study in any respect. There are a bunch of further differences between the study in the mock-up and the study in a real shopping mall: In particular, a real shopping mall allows real shopping! That is, the shopping situation is realistic in that the subject feels like a real shopper; the subject's motivation to use the shopping guide can be increased by a realistic reward scheme in that the subject can keep the items that he was able to buy; the topology of a real shopping mall is different from the topology of the mock-up in a computer science building, in particular, the shopping mall is bigger and more complex than the mock-up that we were able to establish in the first user study.

7.3.1 Realization of the Improved Shopping Guide

Apart from an improved service, the realization of the shopping guide used in the second user study includes two important differences compared to the realization of the shopping guide in the first user study: (1) The use of infrared beacons to localize the user of the system in the first study is simulated by a second handheld device operated by the experimenter and (2) the decision-theoretic model of the new shopping guide takes a fixed deadline into account, which is more in line with the airport shopping example of Chapters 4 and 5.

7.3.1.1 Hardware

In this study, we use two Compaq iPAQ 5450 Pocket PCs. Both handheld computers run the Microsoft Pocket PC 3.0 operating system and are equipped with a build-in *Wireless Local Area Network (WLAN)* card to establish a radio link between the two devices. The subject's handheld computer hosts the actual shopping guide—in principle an improved version of the software used in the first user study. The handheld computer operated by the experimenter hosts a *simulator of the localization infrastructure*.

The experimenter uses the handheld computer to simulate the localization infrastructure in that he sends a radio signal to the subject's handheld device whenever the subject approaches a place where an infrared beacon would be installed in a fully equipped environment. We call such a radio signal a *virtual beacon*. The radio signal is interpreted by the subject's handheld computer (the actual shopping guide) exactly in the way that a signal from an infrared beacon is interpreted in the first study. Figure 7.6 shows the hardware used in the study and a part of the shopping mall map including positions of some of the virtual beacons. The application of this hardware setting is described in the corresponding paragraph of Section 7.3.2.

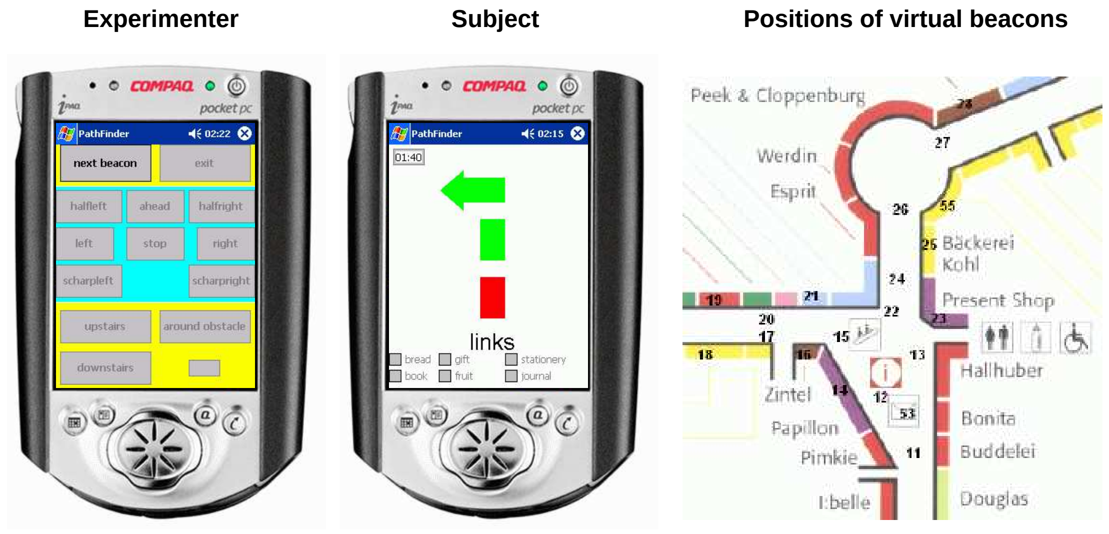


***Figure 7.6**: Hardware used by the experimenter and the subject; positions of virtual beacons.* *(The two PDAs of the experimenter and the subject are connected via wireless LAN. The experimenter can invoke a virtual beacon by pressing a button on his display. Thereby, he simulates the signal that would otherwise be transmitted by an infrared beacon in a fully equipped environment and causes the subject's PDA to display the right navigation recommendation according to the policy. The right-hand part of the Figure shows the positions of some virtual beacons.)*

7.3.1.2 Decision-Theoretic Model

The structure of the Markov decision process used to compute the policy for the study at hand is in principle the same as the structure of the MDP applied in the first user study described in Section 7.2.1.2 (cf. Figure 7.4). However, instead of computing the infinite-horizon policy (which was appropriate in the first user study where we have assumed that time is a pure cost factor), we now compute finite-horizon policies in the way that we have described in Section 4.4.

The decision-theoretic model does not consider that the time to move through the aisles of the shopping mall or to stand in line at the cash desk might depend, for example, on the day of the week or the time of the day.

7.3.2 Second User Study: Method

The overall purpose of the second user study was to confirm under realistic conditions the shopping guide's objective effectiveness and subjective acceptability, which were indicated in the first user study based on the mock-up of a shopping mall (Section 7.2). We do not further question the underlying localization technology in the study at hand but instead focus on the acceptance of the improved (with regard to the system used in the first study) decision-theoretic PDA-based shopping guide in an authentic shopping environment.

User Study with Simulated Infrared Localization In the first place, the user study conducted in the mock-up of a shopping mall investigated the basic usefulness of the decision-theoretic planning approach. In addition, it showed the appropriateness of the hardware solution and in particular of the localization technology on which the shopping guide is based. While it was reasonable to check in the first study if the complete solution that we propose is feasible, useful, and accepted, we do not question the localization technique in the second user study anymore. We are convinced, that the localization technique is in principle appropriate—especially after the transmission range of the infrared beacons has been extended. Therefore, the experimenter takes on the task of the localization infrastructure in the real shopping mall. That is, we have not equipped the shopping mall with infrared beacons but instead the experimenter simulates the signals that would otherwise be transmitted by such beacons.

In order to represent a one-to-one mapping of the infrared localization service provided by real beacons, the reception of the radio signal would have to be dependent on the orientation of the subject. This can only be approximated in a study with simulated infrared localization. The experimenter can only estimate when the subject enters the transmission range of a virtual beacon Moreover, the experimenter will not be able to take the exact angle between the infrared interface of the shopping guide and the simulated infrared beacon into account to decide at what exact point in time to send the radio signal. Insofar, the technically feasible infrared localization method is idealized, but especially in the light of improved infrared beacons can be assumed to be approximated to a realistic extend.

We have tried to facilitate the simulation of the infrared beacons by the experimenter as much as possible, such that this task is not very demanding and is not distracting the experimenter himself. As the experimenter is not supposed to spend much effort on the simulation of the localization infrastructure, the system assists him with this task in the following way: Both, the information about the shopping mall topology (i.e., the locations of the virtual beacons or, more precisely, the mapping from locations to beacon IDs) and the current policy are known to the part of the system running on the handheld device of the experimenter. The experimenter needs to have a good notion of where in the shopping mall real infrared beacons would be installed. When a subject reaches one of the locations where a beacon would be installed, the experimenter only has to press the button “next beacon” to transmit the right signal—this is shown in Figure 7.7 (a). The system knows the initial position of the subject. From the information about the shopping mall topology, the policy, and the feedback given by the experimenter about where the subject walks, the system can always compute which beacon the subject will reach next. From the buttons on the handheld device of the experimenter, only those are activated that make sense in a particular situation. For example, if the subject can only go straight ahead or left, only the buttons to give feedback if the subject has gone straight ahead or left are activated—this is shown in Figure 7.7 (b). If the experimenter has given feedback about the direction to which the subject has gone to, only the button to indicate that the subject has reached the next simulated beacon is activated. As the system provides no navigation assistance within a shop, the experimenter has to give feedback when the subject leaves a shop, such that the shopping guide can take up navigation assistance again—this is shown in Figure 7.7 (c). In a fully equipped environment, this feedback would we realized by an infrared beacon at the shop exit.

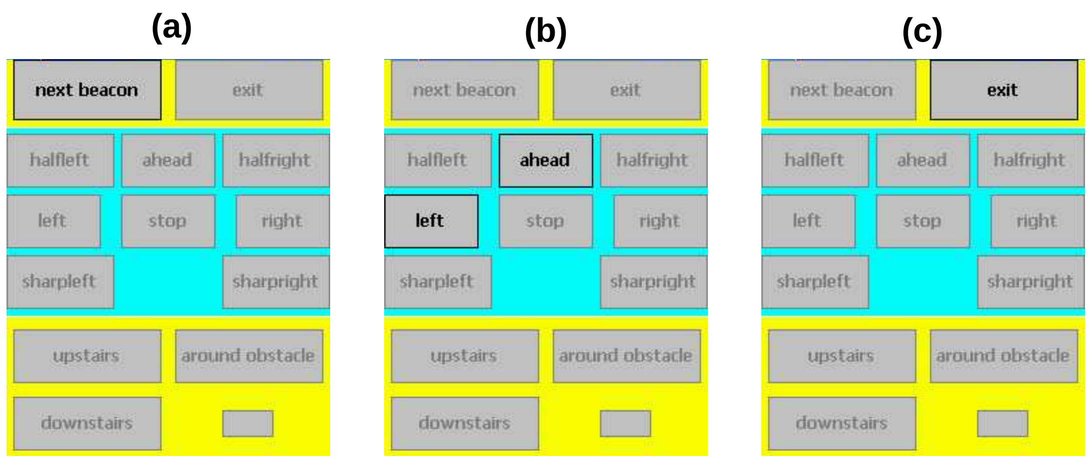


***Figure 7.7**: Graphical user interface on the Pocket PC by which the experimenter simulates infrared beacon signals.*

(The two buttons in the top row are used by the experimenter to indicate that the subject has reached the next beacon (left) or the exit of a shop (right). With the rest of the buttons, the experimenter provides the feedback on how the subject walked. Only valid directions are highlighted. The experimenter can, e.g., give the following feedback: (a) Subject has reached next beacon, (b) subject went left or straight ahead at a beacon, and (c) subject has completed the shopping in a particular shop and reached the shop exit, i.e., the system can continue to give navigation recommendations.)

Shopping Task As in the first user study, each subject is asked to buy a number of items (some bread, a book, a gift item, some fruits, a journal or magazine, and some stationary) from a shopping list. The results of the first study give reasonable reassurance about the principle competitiveness with (if not superiority over) conventional navigation aids such as a paper map. Unlike we did in the first user study, we therefore do not compare the results of using the PDA-based shopping guide in the realistic environment with the alternative use of a paper map in the second user study.

In contrast to the first user study, the buying of the individual products is not simulated in the study at hand. Instead, each subject has a fixed maximum budget (25 Euro) to actually buy one product from each of the six categories listed above—and keep them. To make the shopping as appealing as possible for the subjects, we asked each subject to specify a more or less concrete item for each category. This could, for example, be an item which the subject wanted to buy anyway in the near future. For example, in the category of newspapers and magazines, the subject could specify his favorite weekly magazine, or in the category of bread, he could specify a particular type which he usually buys. This reward policy is supposed to make the shopping task as authentic as it can be and increase the subject's motivation to behave as natural as possible.

The navigation support of the shopping guide ends at the entrance of the individual shops. That is, within a particular shop, the subject himself has to locate the appropriate shelves to find a particular shopping item. Once the subject has found an product, the experimenter checks if the product corresponds to what the subject has specified beforehand and if the price of the selected item is in the allowed range. The maximum budget for the complete shopping task is divided into realistic portions (e.g., 2 Euro maximum to buy a bread, 9 Euro maximum to buy a book). If the subject has selected a valid item, then the experimenter pays for the shopping item at the cash desk. The reward policy enforces that the subject's total reward for taking part in the study is the higher the more products he manages to buy. The difference of the maximum budget and the value of the products that he eventually was able to buy was not payed off.

An additional difference with regard to the first study is a fixed deadline for the shopping to be finished. While in the first study, time was considered as a cost factor (i.e., for each second that the subject shopped, the reward was reduced by a small amount of money), in the second study, each subject was allowed to shop at most 20 minutes. This was explained to the subjects in analogy to having an important appointment (or even the closing time of the shopping mall[10]).

Subjects We recruited 21 subjects (10 of them female). The subjects' age ranged from 21 to 73 years (with an average age of 33.6 and the median age of 27 years[11]). 14 subjects have not visited the shopping mall in which the user study was conducted within the 12 months before the user study, 7 subjects have visited the shopping mall one to three times within the 12 months before the user study. 6 subjects had experience with a navigation system, most of them with a car navigation system. Most of the subjects had experience with a PC, but only two have used a PDA before. Figure 7.8 summarizes the subjects' experiences.

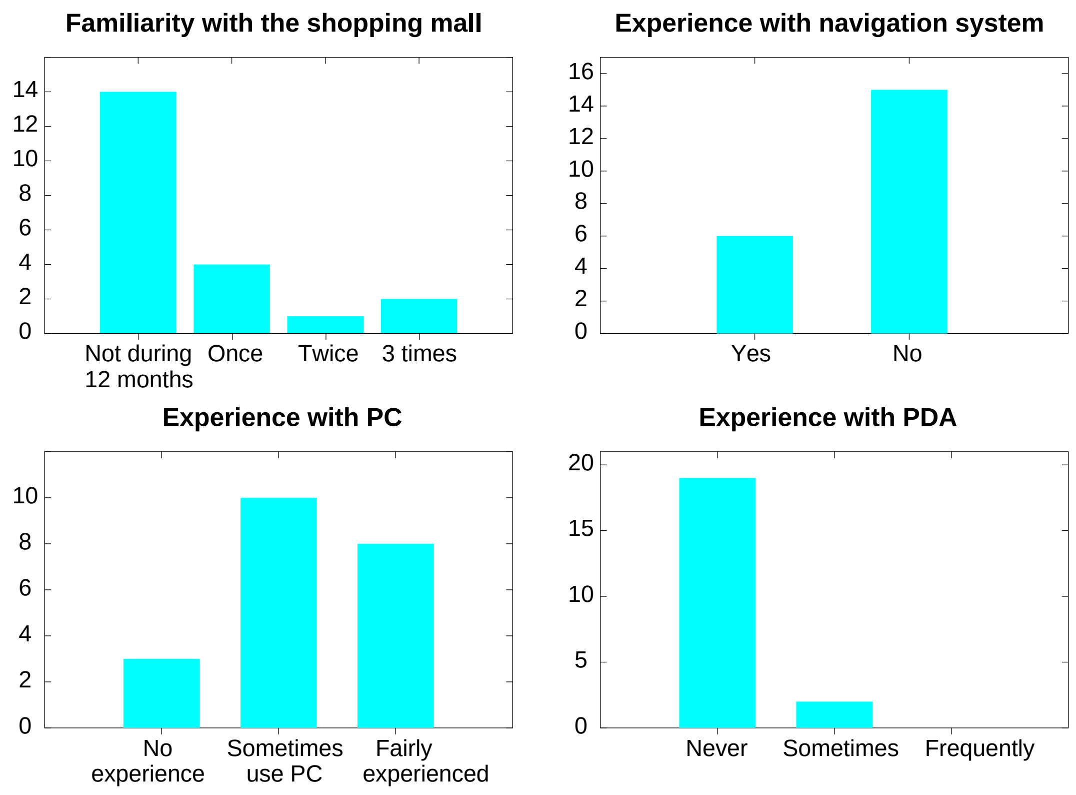


***Figure 7.8**: The subjects' familiarity with the shopping mall and their experiences with navigation systems, PCs, and PDAs.*
(We tried to fi nd in particular subjects who were not familiar with the shopping mall in which the shopping guide was tested. 6 subjects had experience with a navigation system. Most of the subjects have used a PC from time to time or even frequently, but only two have used a PDA before.)

[10]This model also fi ts well with the airport scenario, where the user of a shopping guide must be at the gate in time.
[11]Median age 27 years means: The age of 50% of the subjects was less or equal 27 years

Procedure Each subject performed the task individually together with an experimenter. First, the experimenter explained the nature of the tasks. He then familiarized the subject with the usage of the PDA guide (e.g., what type of information is presented on the PDA display, how to interpret the icons on the overview maps, what the audio signal means, how to check off an item that the subject has bought, and also how the simulated infrared localization technology works). The subjects were explained the reward policy described in the paragraph **Shopping Task**.

In each trial, the subject begins at the mall entrance and is followed around by the experimenter. Each time the subject comes close to a place where an infrared beacon would be installed if the shopping mall was fully equipped with the infrared localization technology, the experimenter simulates the corresponding beacon by pressing the button “next beacon” on the display of his PDA. The virtual beacon is received by the shopping guide and induces the navigation recommendations according to the policy. When a subject enters a shop, the navigation assistance stops and the subject takes a look-around in the shop unguidedly. If he finds an item, the experimenter checks if the item is valid according to what the subject has specified before the trial. If the selected item is valid, the experimenter pays the item on behalf of the subject at the cash desk. The subject checks off the item in the shopping list displayed on the PDA screen. If no appropriate item is available in the recommended shop, the subject moves on to the exit of the shop. At the exit, the experimenter triggers the system to take up navigation assistance again. At this point, the system processes the information whether an appropriate product has been found in the last shop or not.

After the user has visited a number of stores, he will eventually reach the mall exit. At this point, the trial is completed. The subject fills in a questionnaire and is interviewed briefly.

7.3.3 System Behavior

Figure 7.9 gives a notion of what the navigation with the shopping guide used in the second study looks like. The sequence starts in the situation when the subject has just visited a shop and just arrives at the exit of the shop—Figure 7.9 (a). The subject has not yet bought anything (no check box on the bottom of the display is checked off; all check boxes are disabled). The upper left corner of the display announces the time that has already elapsed during the current trial. The subject is about to leave the shop, facing the main aisle of the shopping mall. Shortly after he has turned left, the first overview map is displayed—Figure 7.9 (b). The map includes the position of the user (light green character on the left-hand side of the display) and the location of the next shop that the user is recommended to visit next (light green dot on the right-hand side of the display). As the subject moves through the aisle and reaches the next virtual beacon, the next navigation recommendation (turn half left) is given—Figure 7.9 (c). Navigation recommendations and overview maps (with updated user position information) alter accompanied by audio signals until the subject reaches the next shop and a picture of the storefront is shown on the display—Figure 7.9 (h). The shop is a stationary shop, thus the check box for stationaries is activated. The shopper finds an appropriate stationary item in the shop, buys it, and checks it off. At the exit of the shop, the subject receives the next navigation recommendation (turn right), which recommends him walk on along the main aisle of the mall—Figure 7.9 (i). All check boxes are disabled again.

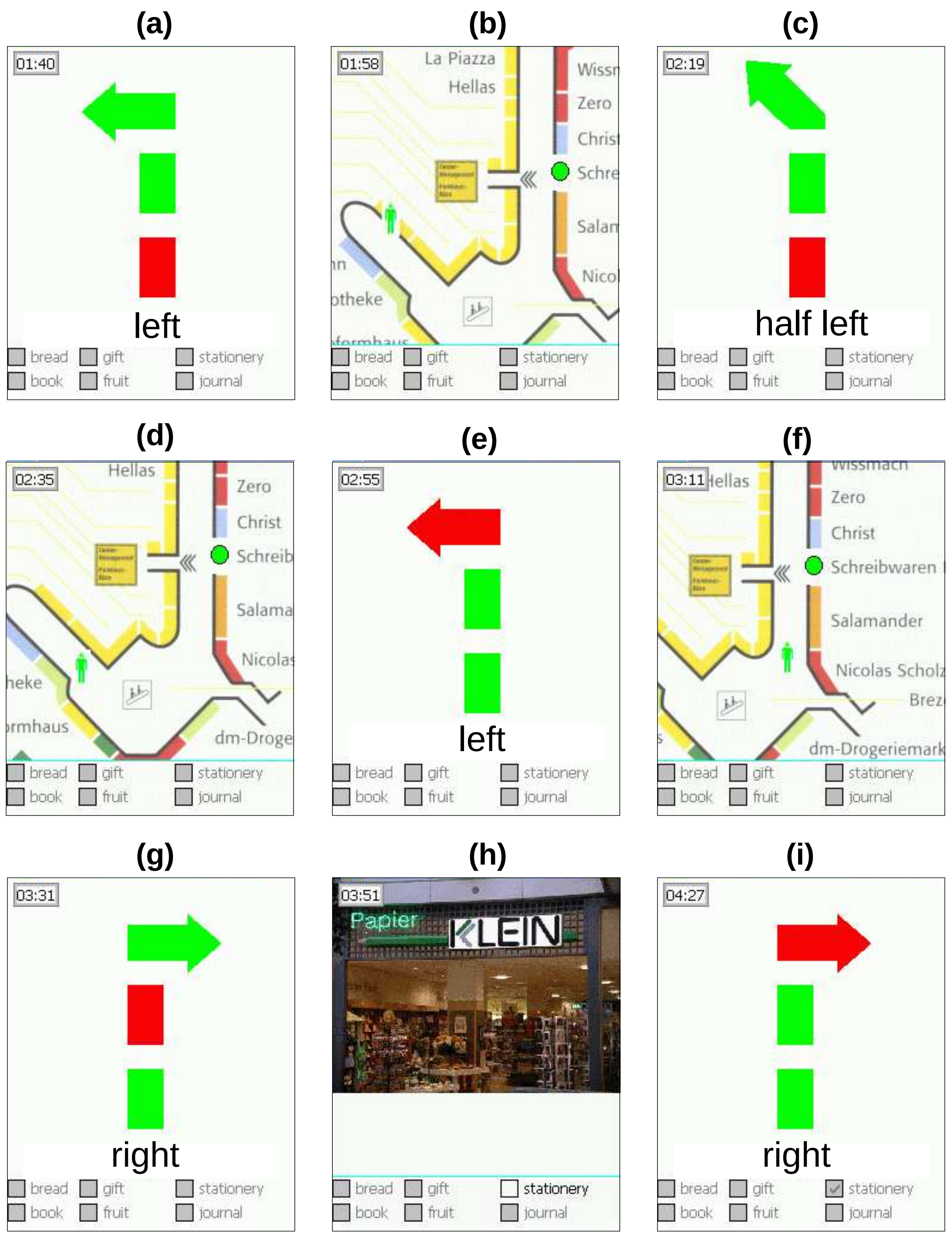


***Figure 7.9**: Nine steps to get from one shop to another.*
(The sequence of screen shots (a) to (g) shows the alternation of navigation recommendations and overview maps that guide the user to the next shop. Once the next shop is reached, the corresponding storefront is displayed (g) in order to facilitate the identifi cation of the right shop. Part (a) like part (i) show the navigation recommendation in the moment when the user is about to leave a shop. Further explanations are given in the text.)

7.3.4 Second User Study: Results

With the second user study, we wanted to find out if the acceptance results of the user study in the mock-up of a shopping mall could be repeated in a real shopping mall. We did not compare the shopping guide with any other navigation aid in the second user study. The study proceeded without problems; the subjects showed considerable interest in the problem and the shopping guide, and they supplied detailed comments on the questionnaire and in oral discussion. In the following, we have analyzed the subjects' performance with the shopping task and the acceptance of the shopping guide in a real shopping mall.

The Subjects' Performance With the Shopping Task Different from the user study in the mock-up of a shopping mall, the shopping guide used for the study in a real shopping mall takes a fixed deadline into account: The subjects had to be at the mall exit within 20 minutes time. To arrange for the subjects arriving at the mall exit in time, we have applied finite horizon decision-theoretic planning as we have described it in Section 4.4.1. The deadline was made by all of the 21 subjects. Most of the subjects managed to buy all 6 items that they had specified before their trial. However, 6 subjects managed to buy only five items, and one subject managed to buy only 4 items (cf. Figure 7.10).

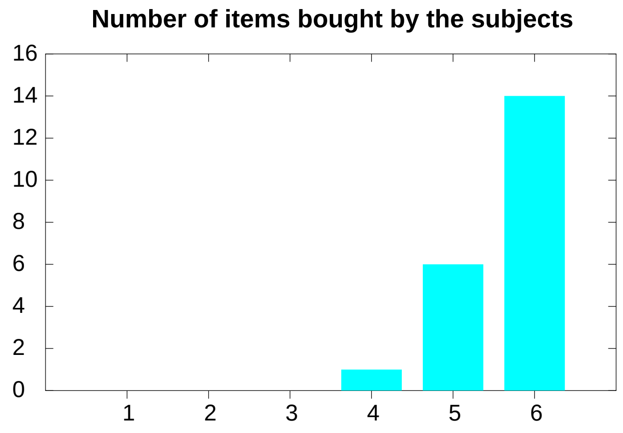


***Figure 7.10**: The subjects' success in completing the shopping task.*
(The shopping list contained 6 items. Most subjects managed to buy all of them and arrive at the mall exit within the time limit of 20 minutes. Those subjects who did not complete the shopping task were recommended by the shopping guide to renounce one or two items from the shopping list in order to arrive at the mall exit in time.)

On average, the subjects spend 18.72 Euro (with a standard error of the mean of 0.8 Euro and a median of 19.35 Euro). The minimum amount spent was 8.36 Euro (the subject managed to buy only 4 items in the given time), the maximum amount spent was 23.92 Euro.

Of those subjects who managed to buy all 6 items, 4 needed between 10 and 15 minutes and 10 needed more than 15 minutes—cf. Figure 7.11 (a). For those 7 subjects who did not manage to buy all 6 items, 5 subjects had between 2 and 4 minutes, and 2 subjects between 4 and 6 minutes left when they reached the mall exit—cf. Figure 7.11 (b).

We have analyzed why it was not possible for 7 of the subjects to buy all 6 items from the shopping list. The subject who managed to buy only 4 items in the given time had slightly less

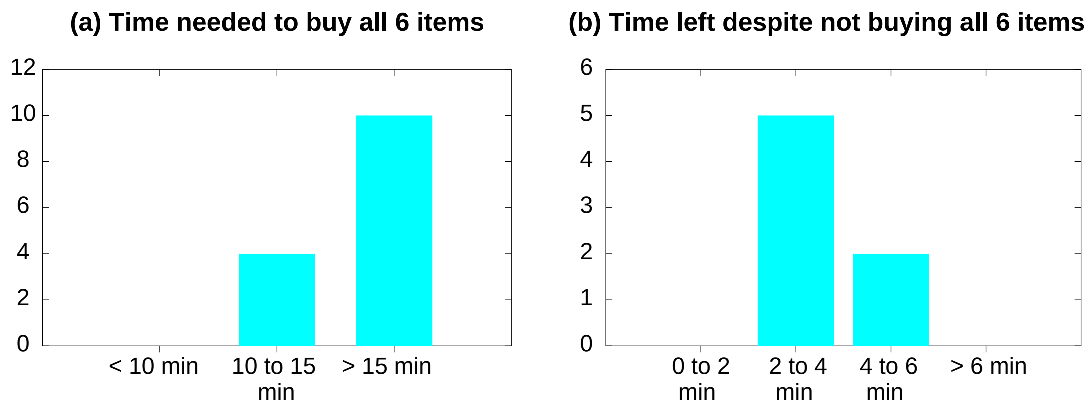


***Figure 7.11**: Time needed for the shopping task.*
(Left: Of those subjects who managed to buy all 6 items from the shopping list, 4 subjects needed between 10 and 15 minutes, which was particularly fast, and 10 subjects needed more than 15 minutes, which was consistent with our expectations. Right: Of those subjects who did not manage to buy all items from the shopping list, 5 subjects had between 2 and 4 minutes, 2 subjects between 4 and 6 minutes left when they reached the mall exit.)

than 6 minutes left when he arrived at the mall exit. However, the two items that he was not able to buy were a book and a gift. For both items, we have assumed that it would already require 5 minutes to find an appropriate item in a corresponding store, plus some time to stand in line at the cash desk—this amount of time is specified stochastically in the domain description. That is, according to the decision-theoretic model, the subject would just have missed the deadline if the shopping guide had recommended to visit a book store or gift shop. Comparing the policy and the log-file assigned to the subject at hand yields that in fact the subject would have been recommended to visit either a book store or a gift shop if he had reached any of the appropriate shops just some seconds earlier. It might be worth mentioning that the subject considered was 73 years old, and although he did not need noticeably more time to move from one location to another, he spend noticeably more time in the shops to compare and find appropriate items. Nevertheless, this subject enjoyed using the shopping guide and could imagine to use it in a similar situation, in particular if he was under time pressure.

For those 6 subjects who managed to buy only 5 items from the shopping list, the shopping guide recommended to renounce the gift item. When they arrived at the mall exit, all of these subjects had only between 2 and 5 minutes left—less than the time assumed for finding a gift item according to the decision-theoretic model. Insofar, it is a desired system behavior if the subject is not recommended to enter the gift shop in favor of getting to the mall exit in time. However, why does the policy not recommend to renounce a less important item (in terms of its monetary value) at an earlier stage of the shopping task (e.g., the magazine) in order to enable the subject to buy a more important (more expensive) item—the gift item—later? In fact, a thorough analysis of the policy yields that the system exhibits exactly this behavior for particular time windows. Yet, if a subject performs well in the early stages of the shopping, it is not necessary to renounce a less important item, as the system's expectation is that the subject will manage to complete the entire shopping task. If the subject then performs worse at a later stage (e.g., needs more time than specified in the decision-theoretic model to find an item in a particular store), then the system cannot help but renounce an item that is supposed to be bought at a later stage of the shopping. Insofar, it is rather unfortunate that the gift shop is close to the mall exit in the user study. However, all of the subjects who managed to buy only 5 items enjoyed using the shopping guide and could imagine to use it in a similar situation, in particular under time pressure or in an unfamiliar shopping mall.

Acceptance of the Shopping Guide in a Real Shopping Mall Figure 7.12 summarizes the feedback that the subjects gave in the questionnaire and that reflects the acceptance of the shopping guide. All subjects enjoyed using the shopping guide, only one subject declared that he enjoyed it just a little. None of the subjects declared that he would not use the shopping guide (provided a fully equipped environment) in a similar situation. With regard to preconditions for which the subjects considered the shopping guide to be particularly useful, 16 subjects mentioned unfamiliar environments, 13 subjects referred to time pressure, and 5 quoted particularly large shopping malls.

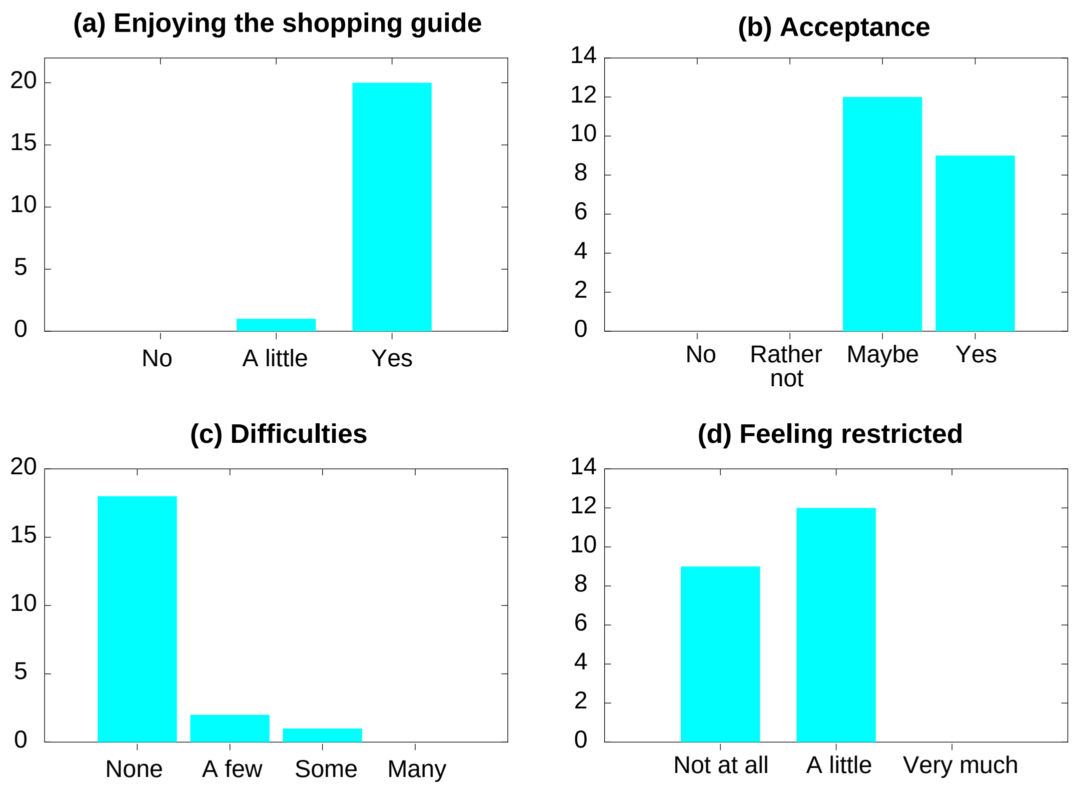


***Figure 7.12**: Acceptance of the improved shopping guide elicited in a user study conducted in a real shopping mall.*
(The subjects answers to the questions: (a) Did you enjoy using the shopping guide? (b) Would you use such a system in an adequately equipped environment? (c) Did you have diffi culties in using the shopping guide? (d) Did you feel restricted by the system?)

Asked about particularly positive aspects about the shopping guide, 9 subjects declared the rapidness of finding adequate shops, 4 subjects mentioned the photos of the shops, which are displayed when a shop is reached, another 4 subjects commended the reliability of receiving the recommendations at the right locations, 3 subjects commended the user-centered recommendations reflected by the arrows on the PDA display, and 2 subjects pointed out the overview maps.

Of those who declare that they would *maybe* use the shopping guide, 3 subjects pointed out that for each recommended store, a selection of alternatives should be presented, another 3 subjects wished that the next item[12] to be bought was displayed, and 2 subjects quoted that changes of the way (initiated by themselves) should be possible.

Only 3 subjects declared that they had difficulties using the shopping guide. One subject mentioned that the guidance was not transparent. He referred to a particular situation, in which the system changed the next shop to be visited along the way to a shop. After the experimenter emphasized in the instructions that the stores indicated in the overview maps of the system reflect only the *expected* next shop to be visited, no other subjects complained about this feature of the system. The system behavior is in fact reasonable (and explainable): If the subject needed less or more time to go from one location to another than it was specified in the decision-theoretic model, then the selected t-time-steps-to-go finite-horizon policy at the next decision point does not need to be consistent with the t-time-steps-to-go finite-horizon policy at the preceding decision point. This might be confusing (because of the obvious incoherence) to the user of the shopping guide if not sufficiently explained in the introduction. For the use of the shopping guide in practice, the system itself should in fact be able to explain such a behavior. This insight stresses the need of an explanation component—a subject which we will study in Section 7.4.

One subject complained that some items (e.g., a book, which could also be considered as a gift item) were not designated to be bought in other shops than those for which the corresponding category was modeled. A more thorough categorization of the items available in each shop will solve this problem. Another subject complained about the feeling of being under time pressure. Although the discomfort of time pressure is understandable, we have explicitly designed the study this way in order to test the finite-horizon decision-theoretic planning approach.

The subjects were also asked if they felt restricted by the use of the shopping guide. Of the 12 subjects who declared that they felt restricted by the system, 8 subjects complained about having to carry the device (i.e., permanently having only one hand free during the shopping), 3 subjects complained about the wire of the ear phone (which could be solved. e.g., by a bluetooth headset), and 4 subjects complained that the shopping guide takes their attention away from the environment or decreases their freedom of decision about their route too much. Although the latter restriction was mentioned explicitely only by 4 subjects, the experimenter noticed indication of this problem in the behavior of other subjects, who did not declare this restriction in the questionnaire, too. An explanation component as we describe it in Section 7.4 might yield relief with regard to this restriction. It seems noteworthy that one subject considered it to be an advantage that the shopping guide takes the attention away from the environment because this way, he was not tempted to buy items that he would not really need.

Suggestions for Improvements of the Shopping Guide The subjects suggested mainly technical improvements of the shopping guide. For example, 3 subjects wished that the device would be smaller. Hardware industry might soon accommodate the users with this desire. Another 3 subjects suggested speech output for the navigation recommendations. Technically it is no problem

[12]The overview maps contained only the information about the next store to be visited, which not necessarily indicated the next item to be bought.

to enable the shopping guide to use speech output. 2 subjects liked the audio signals indicating that the display has changed only if also the way has changed. That is, they did not want to be made aware that they have reached the next location considered in the decision-theoretic model if they are supposed to go on straight, but only if they are supposed to turn. 2 subjects wanted to see a countdown of the remaining time instead of seeing the time elapsed since the shopping started. Another 2 subjects found the directions slightly ambiguous. This problem should be relieved by the use of the real infrared localization technique (as in the user study in the mock-up of a shopping mall). One subject would have preferred vibration alarm instead of (or in addition to) receiving audio signals only. One subject would have appreciated seeing information about queues at the cash desks in the shops. This service requires the consideration of ongoing data from the shops by the shopping guide. And finally, one subject suggested to provide forward-up overview maps. Although this was explicitly declared only by one of the subjects, it could be inferred from how other subjects handled the PDA when an overview map was displayed that they had problems with the orientation of the map, too.

7.3.5 Conclusions

The second user study, conduced in a real shopping mall, has underpinned the feasibility and acceptance of a location-aware shopping guide based on decision-theoretic planning. The first user study was routinely criticized for being conducted in a mock-up of a shopping mall. This criticism was encountered by the second user study.

An important extension of the system used in the second user study was the consideration of a fixed deadline, which involves the application of finite-horizon decision-theoretic planning. Considering that all 21 subjects of the user study have made the deadline (and have managed to buy at least 4 out of 6 items from the shopping list), the second user study has substantiated that finite-horizon decision-theoretic planning is a suitable approach to deal with a fixed deadline in the scenario at hand.

A major drawback mentioned about the system used in the first user study has been corrected to a large extend: The improved shopping guide features overview maps to help the users keep the big picture. Another drawback—the need to attend to the PDA—has not been overcome by the corresponding improvements of the guide used in the second user study and needs further exploration. Moreover, the second user study yielded further suggestions for improvements of the shopping guide, which mainly relate to technical aspects and the usability of the system. Enhancing the shopping guide with an explanation component would be an important step to further increase the usability of the shopping guide.

7.4 Explaining Recommendations

One strength of decision-theoretic planning is that it can take into account the potential consequences of a decision in the future. Decision-theoretic planning trades off the risks and chances by comparing the expected utilities of different options that are available. Suppose, for example, in the airport shopping domain, a user wants to buy a magazine and a book. The system's decision whether to recommend to visit a particular shop A to buy the magazine depends on other possibilities to get the magazine further down the way to the gate, for example, in a shop B or C where at the same time the book can be bought. Decision-theoretic planning can take into account how important the magazine and the book are to the user, at what probability the magazine and the book are available in shop A, B, and C, and even uncertain information about how long the user has to expect to stand in line in a particular shop. However, all this information is cumulated in a single number: the expected utility of a recommendation. Communicating only the expected utility of a recommendation leaves the complexity of the information that yields a particlar recommendation completely intransparent to the user.

In this regard, recommender systems based on decision-theoretic planning are typically black boxes to their users. Similar to how Herlocker, Konstan, and Riedl (2000) have described this flaw in the context of *automated collaborative filtering systems*, the computations that result in a particular recommendation based on decision-theoretic planning are in general far too complex to be understood by the user. Therefore, the upcoming sections explores three questions:

1. What information do the users of recommender systems based on decision-theoretic planning wish?
2. How should appropriate information be presented?
3. How can appropriate information be computed?

7.4.1 Transparency, Predictability, and Controllability

Transparency, predictability, and controllability are among the most prominent usability challenges in HCI. Jameson (2003) associates these challenges with the following questions: (1) *Transparency*: To what extend can the user understand the system's actions? (2) *Predictability*: To what extend can the user predict the effects of his actions? And (3) *Controllability*: To what extend can the user bring about or prevent particular actions or states?

The results of the first user study (see Section 7.2.3.4) have revealed the users' strong desire to keep the big picture when being guided by the PDA-based system (11 out of 20 subject gave comments that pointed into this direction). The desire to keep the big picture relates in to the last two of the three characteristics mentioned above: transparency and predictability.

Wexelblat and Maes (1997) offer the following guidelines to achieve transparency[13] for software agent user interfaces: (1) Use task-specific vocabulary; (2) keep descriptions of the agents' states simple; (3) allow the agent to make functionality suggestions at appropriate times and in a gradual way. In particular, the third advice is important for further improvement of the decision-theoretic shopping guide.

The improvement of the system (used in the second user study) with regard to the ability to provide overview maps helps the user to keep the big picture and enhances the system's transparency and predictability. However, overview maps are good to answer questions like "Which is the shop I am recommended to go next?" or "How far away is the shop that I am recommended to go to next?" But they are not appropriate to answer questions like "Why am I recommended to go to shop X next?" or "Why am I supposed to go left here?" These are questions that explanations generated by the shopping guide are supposed to answer. Apart from an explicit request for explanations by the user, the system can also give explanations automatically, for example, if the user indicates to deviate from the recommendation and if the difference between the expected utility of the system's recommendation and the user's intention exceeds a threshold.

A major challenge will be the exploration of means to enable the system to give useful information at appropriate times, that is, to make sure that the system has the required information

[13]Wexelblat and Maes (1997) use the term *understanding* when relating to transparency.

available when an explanation is requested. To provide appropriate information in a gradual way, the system will have to be able to assess the importance of particular pieces of information for the current user and situation. Herlocker et al. (2000) provide experimental evidence that providing explanations can improve the acceptance of automated collaborative filtering systems. In a similar way, we expect the transparency and predictability of the system behavior that the user gains by an explanation component to increase the acceptance of a recommender system based on decision-theoretic planning.

In Section 2.1.1.1, we have reviewed the Maxims *mail filtering system* with respect to its decision making strategy. The interface agents introduced by Maes (1994) try to increase the transparency of the mail filtering system's internal state via facial expressions that appear in a small window of the user's screen. In the same domain, Pazzani (2000) studies how different representations of user profiles influence the acceptance of mail filtering systems. In an empirical study, Pazzani (2000) finds that some mail filtering profile representations are trusted better than others, although they represent the same content. Similarly, we expect different representations of explanations to be more accepted or less accepted by the users of a shopping guide.

With respect to control, Wexelblat and Maes (1997) recommend to (1) allow variable degrees of control; (2) give users the highest-feasible level of control specification (3) help users understand the agent's model of the user; (4) keep users meaningfully in the loop, particularly for key decision points; and (5) begin with suggestions and move to actions later. For the shopping guide, these guidelines entail that the user should at least at some point have the possibility to make a different decision than the one recommended by the system. On the other hand, allowing the user too much freedom will run the decision-theoretic approach ad absurdum. If the user deviates from the system's recommendations time and again, the policy will most probably have been computed based on incorrect assumptions. However, if the user decides differently than the system recommends in a particular situation because of a specific reason (e.g., the user turns to a not recommended direction to watch the news on a public tv screen for a while but then continues his shopping expedition in accordance to the assumptions that the policy is based on), then the decision-theoretic approach might still be appropriate, even though it might not yield the optimal course of action anymore. Instead of not considering any future consequences of decisions at all, we argue that it is better to plan ahead as far as it is possible, even though we are aware that the system can always be "surprised" by unexpected extraneous events or user behaviors.

The decision-theoretic planning approach is tolerant if the user deviates from the system's recommendations in a way that the user's action is still a *good* action according to the policy. That is, if the user chooses, for example, the second best option (with regard to expected utility) because of a particular reason, then this choice does not violate the key concept behind decision-theoretic planning so much. In particular, if the second best option has an expected utility not much worse than the expected utility of the optimal (with regard to the assumptions made) and recommended action, then future consequences of actions are still taken into account reasonably well and the complete course of action might only be marginally suboptimal. This implies that in particular when explicitly allowing the user for some freedom about decision making, the system should offer options that are compliant with the decision-theoretic approach and suppress those that are not. This way, the user is kept meaningful in the loop, and variable degrees of controllability can be realized: If several nearly equally good options are available, the system can allow for more controllability, while if no reasonable alternative to the recommended actions is available, the system should rather prohibit controllability.

Kay (2001) points out the importance of enhanced controllability for the users of teaching systems in order to increase their motivation and help them to achieve a task effectively. Jameson and Schwarzkopf (2002) discuss pros and cons of controllability in the context of a hotlist for a conference web site. They suggest to check if particular preconditions hold in order to decide if enhanced controllability actually yields the advantage that it may yield in principle. The same holds for controllability of recommender systems based on decision-theoretic planning: The system designer must trade off whether the benefit of enhanced controllability compensates potential suboptimality with regard to the result of the decision-theoretic planning.

7.4.2 Explanations of a Shopping Guide's Recommendations

The decision-theoretic planning approach used in the shopping guide bases its decisions on a comparison of the expected utilities of available options. However, although this is a neat criterion for making rational decisions, it is not very meaningful as a justification for a decision to be understood by a human user. For example, if the user asks why the system prefers to recommend turning right over turning left, he will usually not be satisfied if answered that the expected utility if he turns right is 10.7, while the expected utility if he turns left is only 6.8. Expected utility is not useful as the basis for generating *explanations* for decisions based on a decision-theoretic planning approach. Instead, a shopping guide might present a comparison of the available options as indicated by Figure 7.13 (b).

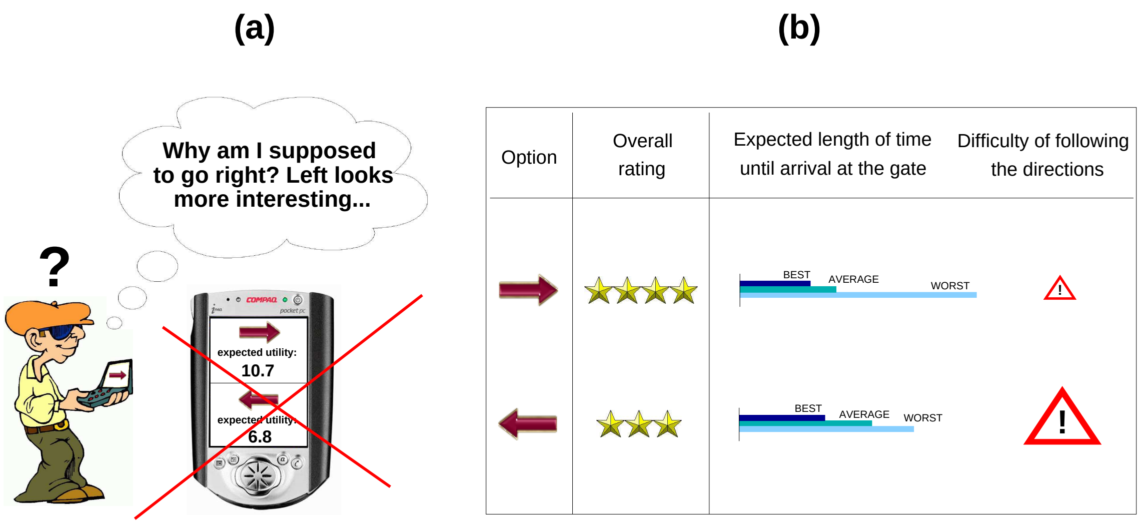

(a)
(b)
Why am I supposed to go right? Left looks more interesting...
?
COMPAQ
expected utility: 10.7
expected utility: 6.8
Option
Overall rating
Expected length of time until arrival at the gate
Difficulty of following the directions
BEST
AVERAGE
WORST
BEST
AVERAGE
WORST

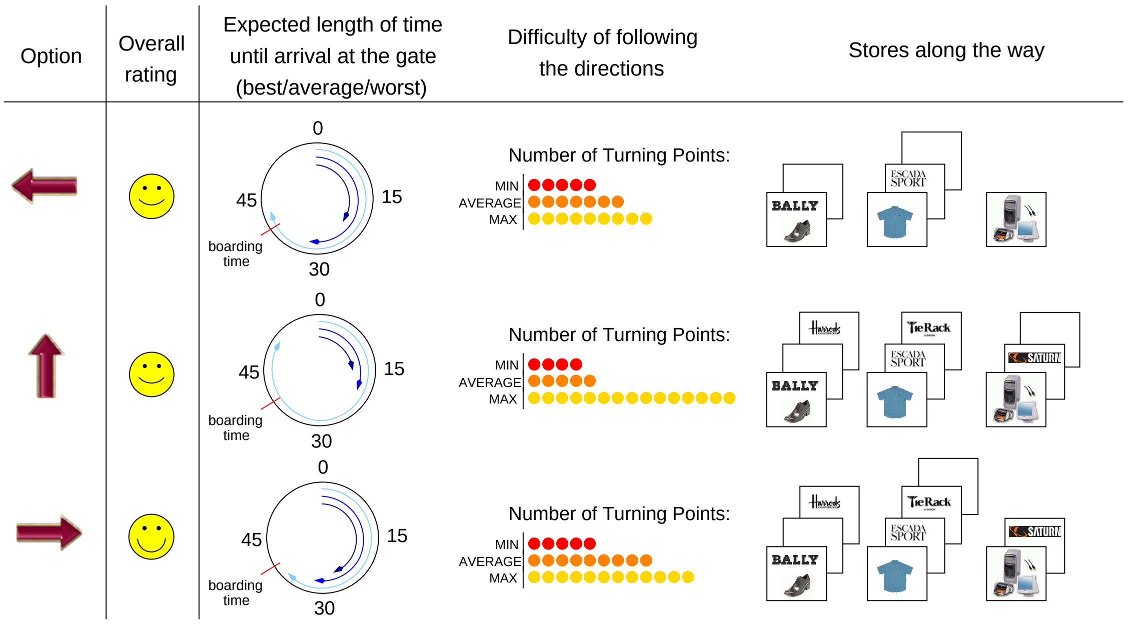
***Figure 7.13**: Explaining navigation recommendations—motivation.*
(Expected utility (a) is not convenient to justify or explain a recommendation given on the basis of decision-theoretic planning. (b) Instead, the system should come up with information like: an overall rating for a recommendation, the expected time for finishing a task, and the difficulty of following upcoming directions.)

7.4.2.1 User Study: Method

We have developed a questionnaire to study (1) which information potential users of a shopping guide consider to be useful in terms of explanations of decisions made by the system and (2) what users consider to be appropriate representations for such explanations. The questionnaire was filled out mainly by third year students of a lecture on *Intelligent Environments*[14].

Materials The questionnaire comprises 6 pages. The first part of the questionnaire comprises 4 pages, which are concerned with the question about what information potential users of an airport shopping guide find helpful as explanations of decisions made by the system. The second part comprises 2 pages, which are concerned with the appropriateness of different forms of representation for recommendation explanations.

The **first part** of the questionnaire has two subparts. The first subpart—we refer to it as part 1.1—involves three tables, each with assessments of three optional routes (see Figure 7.14). These tables were supposed to help the subjects familiarize with the kind of recommendation explanations that we had in mind. Each table gives an overall rating for each of the optional directions. Moreover, for each optional direction, it contains information about the expected duration of getting to the gate (best, worst, and average duration), an assessment of the difficulty of following upcoming navigation recommendations, and eventually information about shops along the way.



Option	Overall rating	Expected length of time until arrival at the gate (best/average/worst)	Difficulty of following the directions	Stores along the way
←	☺	0, 15, 30, 45, boarding time	Number of Turning Points: MIN ●●●●● AVERAGE ●●●●●●● MAX ●●●●●●●●●	BALLY, ESCADA SPORT
↑	☺	0, 15, 30, 45, boarding time	Number of Turning Points: MIN ●●●● AVERAGE ●●●●● MAX ●●●●●●●●●●●●●●●	Harrods, BALLY, TieRack, ESCADA SPORT, SATURN
→	☺	0, 15, 30, 45, boarding time	Number of Turning Points: MIN ●●●●● AVERAGE ●●●●●●●●● MAX ●●●●●●●●●●●●	Harrods, BALLY, TieRack, ESCADA SPORT, SATURN

***Figure 7.14**: Example of a table describing three navigation options.*
(For each of the three options, the table gives an overall rating and compares some characteristics of each of the options. To represent the characteristics, different representations were used in the other two tables—see Appendix H. The subjects were asked to select the option that appeared most attractive to them and give comments on why they have decided for this option.)

We have altered the representation of particular information and also the degree of detail of particular types of information among the three tables. For example, the expected duration until arrival at the gate is once represented with bars and once with a clock-like diagram, and it includes

[14]The lecture was given by my colleagues Andreas Butz and Antonio Kr¨uger in the winter term 2002/03.

information about the boarding time or not. In this part of the questionnaire, the subjects were asked to choose from each table the option that appears most attractive to them and explain why they have decided for the option. For each table, there was half a page of space for the explanation of the subject's selection and for comments of any kind.

The second subpart—we refer to it as part 1.2—summarized the three tables of the first subpart and the students were asked if some of the information used in the three tables (i.e., overall rating, duration until arrival at the gate, difficulty of following upcoming recommendations, and information about shops along the way) can be left out. The subjects assessed the usefulness of each type of information (independent of how it was presented in each particular table) by checking off one of the following statements: (1) Better to leave it out; (2) equally good left out or included; (3) can be left out if necessary, but better kept; (4) should definitely not be left out.

The **second part** of the questionnaire analyzes which of the forms of presentation that were used in the first part of the questionnaire have been most comprehensible to the subjects. For each type of information, three different forms of presentation were used.

(a) **Overall rating:** (1) Smiley representation; the more happy the smiley looks like, the better is the assessment of the option; we decided to use the smiley representation in the questionnaire because of the assumption that it can at least roughly be understood without any accompanying commentary; we assumed five grades of the facial expression to be easily distinguishable; (2) textual representation; one or two words describing the quality of the option, for example, "recommended" or "highly recommended"; similar textual assessments can be found in magazines testing and comparing technical devices; (3) star rating; a number of stars (0 to 5) reflects the quality of an option; this kind of representation is well known, for example, in the context of movie ratings. The three representations are shown in Figure 7.15 (a).

Another option to represent an overall rating would have been a mark as it is used in the school system (e.g., A level, B level, and so forth, as in the American and British school system, or a number from 1 to 6 or from 00 to 15, as in the German school system). However, we did not find this representation intuitive enough.

(b) **Expected length of time until arrival at the gate (best/average/worst):** (1) Bar representation of best, average, and worst duration, but no information about how the durations relate to the boarding time; (2) bar representation of best, average, and worst duration, including information about how the durations relate to the boarding time; (3) clock-like representation of best, average, and worst duration, including information about how the durations relate to the boarding time.

By leaving out the information about the boarding time in the first representation, we wanted to find out if people automatically assume that the boarding time has been taken into account even if not being represented explicitly, that is, if they feel like they do not have to care about getting to the gate on time if the boarding time is not mentioned in the histogram. Option (3) was supposed to elicit if the same information that is given in option (2) is more comprehensible if the representation resembles a clock, that is, if the representation is somehow associated with the concept of time. The three representations are shown in Figure 7.15 (b).

(c) **Difficulty of following the directions:** (1) Caution sign representation; the size of the sign corresponds to the difficulty of following upcoming recommendations; we assumed that people from most cultures, especially if they drive a car, are familiar with the caution sign and interpret it in a reasonable way; (2) minimal, average, and maximal number of turning points represented in one row; (3) minimal, average, and maximal number of turning points represented in three rows. The three representations are shown in Figure 7.15 (c).

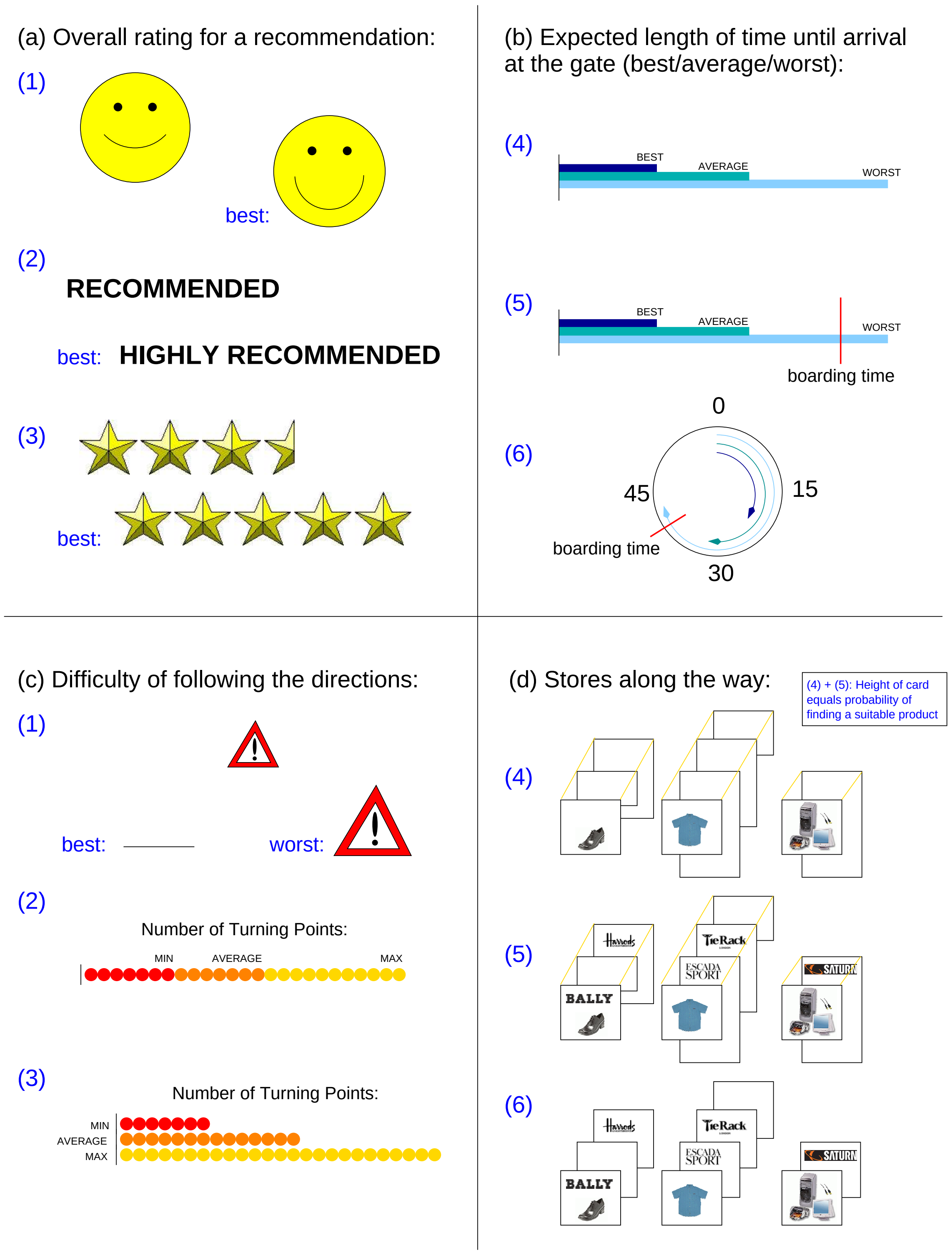

(a) Overall rating for a recommendation:
(1)
best:
(2)
RECOMMENDED
best: HIGHLY RECOMMENDED
(3)
best:
(b) Expected length of time until arrival at the gate (best/average/worst):
(4)
BEST
AVERAGE
WORST
(5)
BEST
AVERAGE
WORST
boarding time
(6)
0
15
30
45
boarding time
(c) Difficulty of following the directions:
(1)
best:
worst:
(2)
Number of Turning Points:
MIN
AVERAGE
MAX
(3)
Number of Turning Points:
MIN
AVERAGE
MAX
(d) Stores along the way:
(4) + (5): Height of card equals probability of finding a suitable product
(4)
(5)
(6)
Harrods
TieRack
ESCADA SPORT
BALLY
SATURN

Figure 7.15: Different representations of the overall rating for a recommendation, information about the expected time to get to the gate, the difficulty of following upcoming directions, and information about stores along the way.
(Explanation in text.)

As an alternative to the discrete representations (2) and (3), we initially thought of a continuous representation like the one for time until arrival at the gate. However, if the difficulty of following the directions is related to discrete events, for example, the expected number of turning points, then the discrete representation is more reasonable. If the measurement of the difficulty of following the directions additionally includes other criteria (e.g., the number of possible directions at turning points and the expected amount of pedestrian traffic along the way), than a different, possibly a continuous representation might be more appropriate.

(d) **Stores along the way:** (1) Information about categories of stores along the way (presented on small cards) and estimations of the likelihood (which corresponds to the height of a particular card) to find an appropriate product in any particular store; (2) information about concrete stores along the way and estimations of the likelihood to find an appropriate product in each particular store; (3) information about concrete stores along the way, but no estimations of the likelihood to find an appropriate product in any particular store. The three representations are shown in Figure 7.15 (d).

The representation of information about stores along the way can be arbitrary complex. Even trying to capture just some of the useful information about stores can result in a rather complex representation. The cards, which we used in all three variations, allow to represent information compactly by overlappings and can be further refined. It is possible to accommodate more information in the cards, for example, use the size of the icons and labels to indicate the probability that the user will actually get to a particular store. Or the cards might include information about opening hours, the expected time to stand in line at the cash desk, or details about the products sold in a particular store—viewing such kind of information might involve zooming into a card. There seem to be so many possible variations of this kind of information that these variations might well be the subject of a user study of its own.

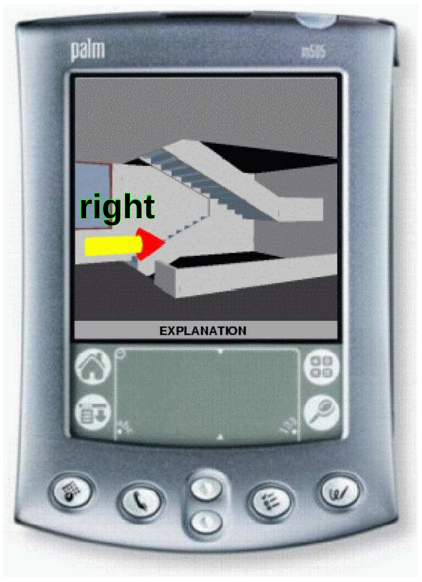
Each subject assessed each of the three different forms of presentation for each type of information on a scale with seven grades ranging from "not useful at all" to "very useful". The complete questionnaire is summarized in Appendix H.

Design The questioning was designed as a within-subject study. All subjects were asked to complete the same questionnaire under the same conditions.

Subjects We asked 33 subjects to fill in the questionnaire. All subjects were students in a computer science lecture on *Intelligent Environments*. Some of the students were neither German nor English native speakers, but all of them spoke English well enough to follow a computer science lecture held in English. Therefore, we assume that all subjects were able to follow the instructions and understand the questionnaire.

Procedure All subjects were jointly instructed at the beginning of the lecture. The experimenter asked the subjects to imagine that they would be using an airport shopping guide: a system that guides the user to the gate and assists him in locating appropriate stores for their shopping on the way to gate. The subjects were already familiar with the kind of system that they should imagine to use, because similar systems and their underlying technology have already been introduced and explained in preceding lectures. The experimenter gave a survey of the airport shopping guide and its underlying technology to refresh the subjects' knowledge.

Then, the experimenter put slides on an overhead projector. The first slide showed the shopping guide screen depicted in Figure 7.16 (left-hand side), with a PDA giving the navigation recommendation to turn right. In this situation, the subjects were asked to imagine that they wanted to know why they are supposed to turn right—although some other direction might look more interesting to them. The introductory slides were also used to explain each individual page of the questionnaire at length.



- You want to know why the system gives this recommendation.
- Therefore, the system should be able to give you an explanation.
- Questionnaire to find out what kind of explanations are useful.
- For the questionnaire, assume that you have no recommendation by the system but just an assessment of available options!

Figure 7.16: *Part of the materials shown on an overhead projector while the experimenter instructed the subjects.*
(Left: The display of the pretended shopping guide depicts the situation which the subjects were asked to imagine. Right: Extracts of the slides which the experimenter used to explain the instructions for fi lling out the questionnaire.)

After having given the instructions, the experimenter handed out the questionnaire (surveyed in the paragraph **Materials** and described at length in Appendix H) with questions on the usefulness of particular pieces of information for explanations and forms to present the information. The subjects were asked to take up to about 20 minutes to complete the questionnaire and pose questions at any time if anything had remained unclear during the instructions or in the questionnaire.

7.4.2.2 Results and Discussion

The questionnaire was designed on the basis of a particular recommender system based on decision-theoretic planning in the airport shopping domain. It gives answers to two questions: (1) What information is desired as explanations? And (2) how to present explanatory information? Some of the comments given by the subjects provide interesting feedback that relates to the concept of controllability.

In the first instance, part 1.1 of the questionnaire was supposed to help the subjects to get familiar with the situation that they were supposed to imagine and the kind of explanations that we had in mind for the recommendations given by the decision-theoretic airport shopping guide. Each of the three tables in part 1.1 was designed in a way that one option appears superior to the other two. However, as Figure 7.17 shows, for each of the tables, there were some subjects who did not select the apparently superior option. Some of the comments by which the subjects justified their decision in such cases were surprising to us: For example, one subject holds the view that “a small number of stores makes the selection of a suitable product easy.” Another subject pointed out that he “does not expect to need the worst case time until arrival at the gate”, while missing their flight in the worst case was probably the strongest argument against this option for most of the other subjects. Altogether, part 1.1 shows that the subjects give different priorities to the offered pieces of information. From this result, we infer that if offered controllability, the users of a system like the airport shopping guide would in fact sometimes decide differently from how the system recommends to.

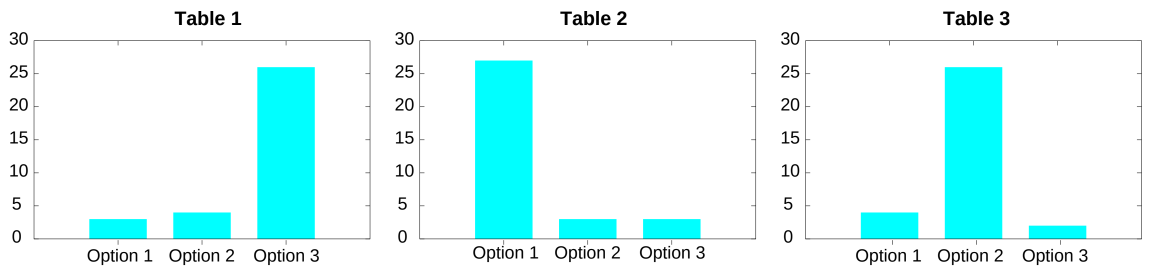


***Figure 7.17**: Results of the questionnaire’s part one, subpart one.*
(In part 1.1 of the questionnaire, for each of the three tables, the subjects were asked to decide which one out of three options characterized in each table they would choose. We designed each table in a way that one option appeared to be superior to the other two. However, for each table, there are a few subjects who decided for one of the apparently inferior options.)

Of those subjects who have selected the apparently superior option, nearly two-thirds referred to the expected time until arrival at the gate to justify their decision. Not surprisingly, most comments bring out the importance of reaching the gate before the boarding. But three subjects also note that they “don’t have to wait too long at the gate” when choosing this option, which indicates that they would feel like wasting time if they would reach the gate (far) too early. The decision-theoretic planning approach is well suited to care about not wasting time waiting while possible secondary goals can still be reached (see Section 4.4).

What Information is Desired as Explanations? The comments given in part 1.1 already anticipate some of the qualitative results of part 1.2, for example, the high importance that is attached to the expected arrival time at the gate. The comments that relate to the information about stores along the way already show that the assessment of the probability to find a suitable product in a particular shop is taken into account by some subjects. The comments that relate to the difficulty of following the directions indicate that, in particular because of being guided by a navigation system, some subjects do not care too much about this piece of information (6 subjects gave comments that indicate this opinion). Finally, with regard to the overall rating (in the case of a textual representation) one subject gave the comment that he has “just chosen HIGHLY RECOMMENDED because there is not much difference in the rest of the information” about the three options. This indicates that an overall rating is rather considered to be the last resource, which is probably because of the marginal explanatory nature of this piece of information.

In part 1.2 of the questionnaire, the subjects were directly asked about the importance that they attach to the four pieces of information they were offered in the three tables of part 1.1. The histograms of Figure 7.18 reflect what the subject's comments in connection with part 1.1 have already indicated: The most important criteria for the subject's selections are the expected time until arrival at the gate and the information about the stores along the way. With regard to the overall rating and the assessment of the difficulty of following the directions, the corresponding histograms show the tendency that the subjects have the feeling that they could well do without this information.

How to Present Explanatory Information? The histograms of Figure 7.19 show the subjects' preferences among the different representations for the explanatory information used in the three tables of part 1.1. We discuss the representation assessments for each piece of information in turn.

(a) **Overall rating:** With regard to the overall rating, the star representation attained the best assessment. The subjects in particular appreciated that the difference between different ratings is easy to recognize (3 comments) and mentioned that they know this representation from other assessments (3 comments). The representation was characterized as precise, useful, compact, clear and easy to compare. Mentioned drawbacks (e.g., the difficulty to distinguish between good an best) can easily be eliminated.

The overall rating represented by smileys or textually still received good assessments. Pros mentioned about the smileys are, for example, that they are intuitive and universally understood. However, the mentioned cons indicate that the different ratings are difficult to distinguish (more than a third of the subjects gave comments pointing to this drawback). The smiley representation was also characterized as ambiguous, too imprecise and childish. In addition to difficulties with regard to the graduation of the textual representation (3 comments), two subjects pointed to the (obvious) drawback of language specifity. Another two comments indicated that the subjects just prefer icons/pictures to text.

(b) **Expected length of time until arrival at the gate (best/average/worst):** With regard to the expected length of time until arrival at the gate, the subjects considered it to be crucial to see this information in relation to the boarding time. The histogram that says nothing about how the expected time until arrival at the gate relates to the boarding time received the worst assessment—15 subjects complained about this shortfall. The histogram that includes the boarding time is assessed better than the clock-like representation that includes the boarding time.

The histogram including the boarding time was characterized as clear, well structured, and easy to understand. One drawback that was mentioned is the lack of a time scale (4 comments). The clock-like representation also received a rather good assessment. Similar to the histogram including the boarding time, it was characterized as exact, clear, and quickly understood. However, some comments indicate that in a direct comparison, the clock-like representation is not as easy to understand as the histogram representation (both including the boarding time). The comments of 7 subjects pointed into this direction. Some subjects found the clock-like representation too complex, unclear, or confusing. A few subjects felt that the clock-like representation wastes PDA screen space or could be difficult to be recognized on a small display. One subject pointed to problems when the displayed times exceed one hour.

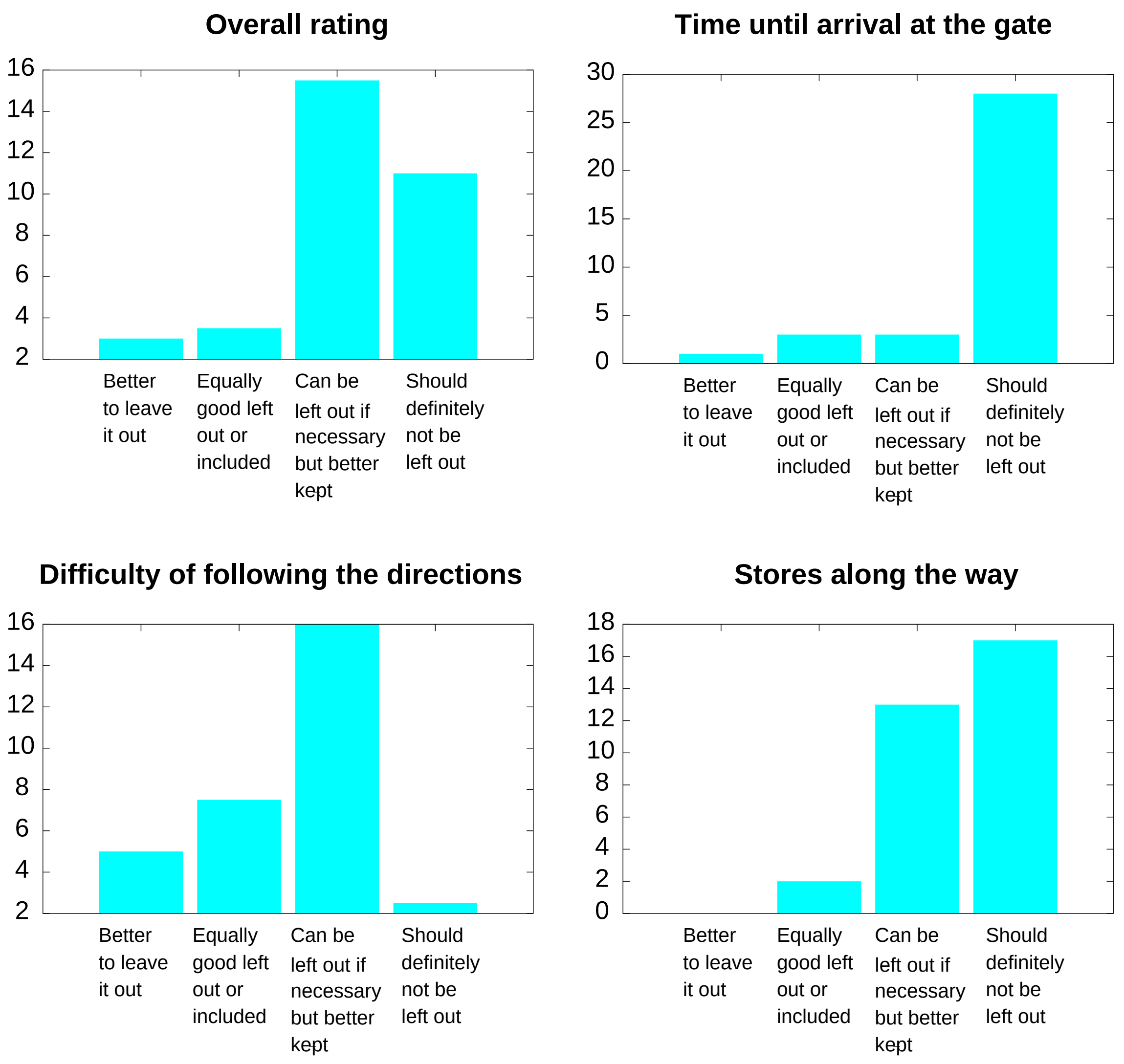


Figure 7.18*: Results of the questionnaire's part one, subpart two.*
(In part 1.2 of the questionnaire, the subjects were asked to assess the importance/usefulness of each piece of information. In the particular domain of airport shopping, an estimation about the time until arrival at the gate was assessed to be highly important. Also information about stores along the way are found very useful. Many subjects felt like being able to do without an overall rating for the different options and also the information about the diffi culty of following the directions was not considered to be essential.)

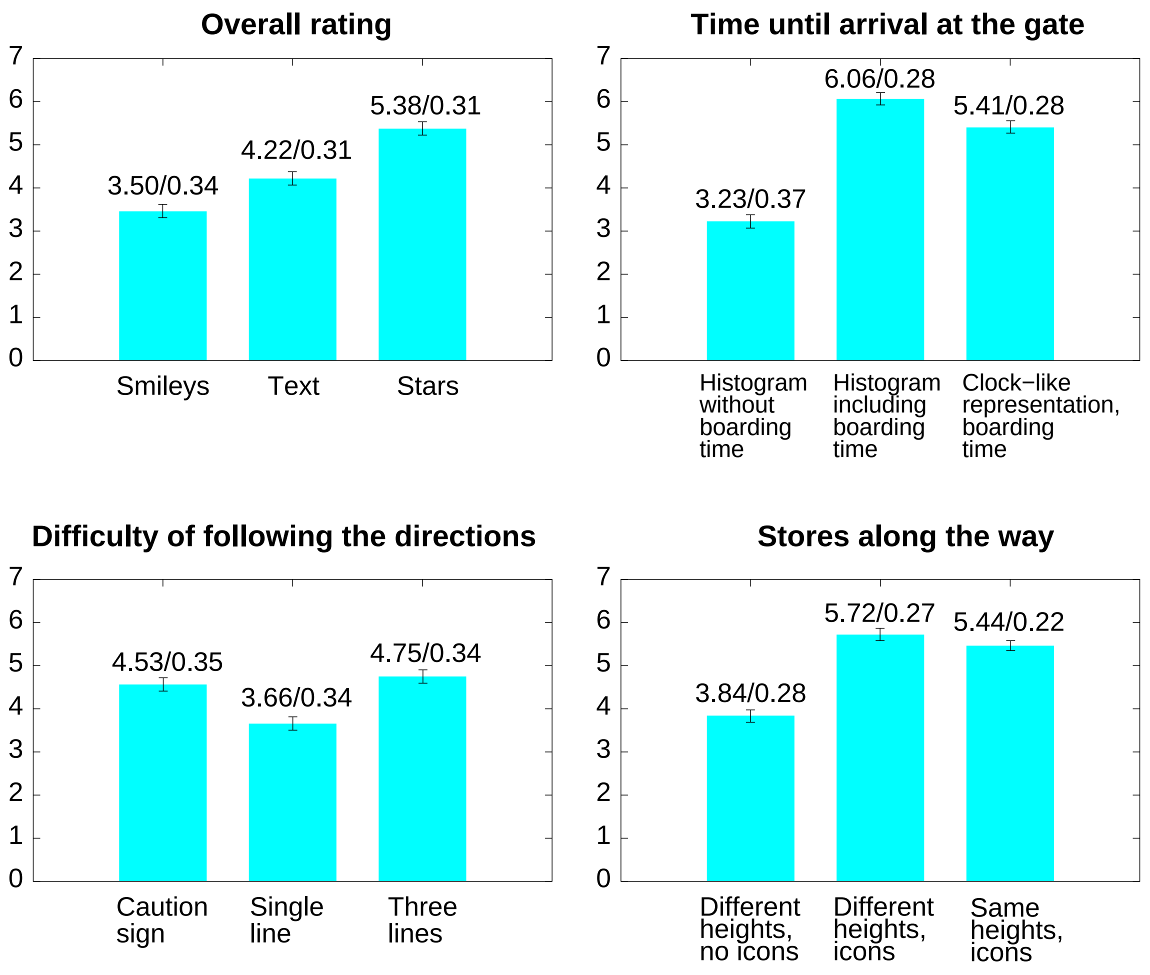


***Figure 7.19**: Results of the questionnaire's part two.*

(In the second part of the questionnaire, the subjects were asked to assess the appropriateness of the explanation representations: For an overall rating, the well established representation by a number of stars was preferred; for the expected time until arrival at the gate, the subjects liked the histogram including information about the boarding time best; for the diffi culty of following the directions, the qualitative description by a caution sign was assessed about as good as the quantitative description in terms of the expected number of turning points; and eventually for the stores along the way, the subjects appreciated the most detailed representation including a precise declaration of upcoming shops and an assessment of the likelihood to fi nd some appropriate item there. The bars in the histograms are labeled with the mean values and the standard error of the mean values. The overlapping intervals of the standard error of the mean in the histogram Diffi culty of following the directions *indicate that the ratings of the* Caution sign *and* Three lines *representations do not differ signifi cantly. The overlapping intervals of the standard error of the mean in the histogram* Stores along the way *indicate that the ratings of the* Different heights, icons *and* Same heights, icons *representations do not differ signifi cantly.)*

(c) **Difficulty of following the directions:** With regard to the representation of the difficulty of following the directions, the assessments of the caution sign representation (pure qualitative information) and the three line representation of the expected number of turning points (including quantitative information) are nearly equal. They both outperform the representation of minimal, average, and maximal number of turning points in one row.

In their comments, the subjects appreciated that the caution signs are clear just at a glance, intuitive, easy to understand and to compare. What is considered as advantage by some subjects (clearness just at a glance) implies a disadvantage for other subjects: The scale is not known, the representation does not give full information, and the signs are imprecise. Two subjects mentioned a conceptual problem with the caution signs: Even a small caution sign is still a caution sign! If the directions will be very easy to follow, these subjects would prefer a thumbs-up sign instead of a small caution sign. For the representation of the difficulty of following the directions in three lines, neither the pros nor the cons mentioned by the subjects were very strong. Two subjects mentioned explicitely that they do not consider this information to be very important. Compared to the same information represented in one line, one subject liked the three line representation better if enough space was available. The compactness of the single line representation was emphasized by the comments of four subjects.

(d) **Stores along the way:** With regard to the information about stores along the way, the representation including shop labels and information about the probability of finding some appropriate product there received the best assessment. The representation that gives information about the probability of finding a suitable product in a particular shop but does not say the name of the shop received the worst assessment. If the representation includes the shop label but gives no information about the probability of finding a suitable product there, then it is not assessed much worse than if the probability of finding a suitable product is indicated by the height of the cards. That is, the precise labels of the shops seem to be preferred to an assessment of the probability to find a suitable product in a shop if the subject does not know which shop it is.

Most of the comments show that the subjects appreciate a high level of detail with regard to this piece of information—which is in contradiction with the findings of (Herlocker et al., 2000), who conclude that people preferred simple explanations. So it may be that simplicity itself is not the criterion but rather: Does the added complexity add really valuable information? 11 subjects gave comments that point to the preference for detailed information. 6 subjects explicitly complained when the names of the shops were missing. A few subjects made general comments like all of the three representations might be too large for a small display (or not recognizable) or that they do not trust probabilities in general.

Two subjects mentioned that the auxiliary lines (cf. Figure 7.15), which are used if different heights of the cards indicate the probabilities to find a suitable product, are useful. One subject proposes to indicate with green and red backgrounds if a shop is open or closed.

7.4.3 Computing Explanations

The histograms of Figure 7.18 indicate the following importance ranking of the types of information that we suggested as explanations for navigation recommendations: Most important to the subjects comparing different options is the information about the expected time of arrival at the gate in relation to the boarding time. This result underpins the need to consider fixed deadlines in the decision-theoretic model as described in Section 4.4. Second most important seems to be the information about stores along the way. This means that the two most important pieces of information relate to the user's primary goal and secondary goals. Slightly less important seems to be an overall rating of alterative options. Eventually, the subjects were discordant about how to assess the usefulness of the information about the difficulty of following upcoming directions of alternative options. The following paragraphs will present approaches to provide the necessary information in the order of how the subjects rated the importance of the particular pieces of information. The implementation of these approaches goes beyond the scope of this thesis.

Expected Length of Time Until Arrival at the Gate (Best/Average/Worst): The expected length of time until arrival at the gate can for each physical location be computed straight from the policy[15] and the domain description (i.e., the topological description of the airport building and where infrared beacons are arranged in the building). We usually declare the distance between two adjacent locations in terms of the time needed to move from one location to another. These measures can initially be estimates (determined by the software designer simply by measuring the time that he needs to cover the corresponding distances). Such estimates can successively be adjusted by an analysis of the data collected during the use of the system.

Determining the best case length of time until arrival at the gate requires only a single simulative execution of the policy. During the simulation, the system assumes that everything works out ideally, for example, that the user never takes a wrong turn and achieves any secondary goal (e.g., buying a product) in the first attempt.

The average length of time until arrival at the gate can be determined by repeatedly simulating the execution of the policy and averaging the length of time needed to reach the gate. The system thereby simulates stochastic actions according to their probabilities. As in the decision-theoretic model, the probabilities applied in the simulations can depend, for example, on the time of the day or the day of the week.

Determining the worst case length of time until arrival at the gate is more difficult than considering the best and average case. If the domain description contains cyclic transitions, the worst case time is in theory infinite. This information is useless as explanation. Instead, the system can provide information about a "soft" worst case, e.g., the worst case that occurred during the simulations for the average case.

It is possible to determine the worst case time if the domain description does not contain any cyclic transitions. The domain description in the model that we employed for the user studies in a shopping mall scenario described in Sections 7.2 and 7.3 were nearly acyclic: By introducing one-way passages, we could guarantee that the user could never come along any physical location more than once. However, the waiting procedure at the cash desks of the shops brings about cycles in the decision-theoretic model. If we chose a different, acyclic model for the waiting procedure, the worst case length of time until arrival at the gate could be computed in analogy to the best case time. During the simulation, the system assumes that everything "goes wrong", for example, that the user always takes the worst wrong turn and never achieves any secondary goal. However, it is doubtful that this procedure really yields information that is useful as an explanation. Instead, strategies similar to the computation of "soft" worst cases seem to be more appropriate.

Stores Along the Way: What stores the user is recommended to go to depends strongly on the results of visiting other stores along the way. If there are several possible stores to buy a particular

[15]When we speak of a policy in this section, we mean both a single infi nite-horizon policy for the case of time being considered as a pure cost factor and a set of fi nite-horizon policies if a fi xed deadline is involved.

item, then those options other than the one with the highest expected utility will only get relevant if the user does not find the item in the first store. This implies the following strategy to give information about upcoming stores: The approach requires one simulative execution of the policy for each item to be bought (or in general, for each secondary goal to be achieved). During the simulation, it is assumed that the user does not find the desired item in any store and thereby checks what other stores that offer the desired item will be recommended in consequence.

As the information is supposed to show the user in how far existing options differ in terms of the upcoming stores, it might not be necessary to simulate the execution of the complete policy. If two alternative routes join again after some navigation recommendations, the comparison of the alternatives can be restricted to the those parts that really differ.

Which stores the policy will recommend for a particular item depends also on what other items the user has already bought (or which other secondary goals the user has already achieved, respectively). For example, if the user has items A and B on his shopping list, then the next recommendation to find item A can depend on whether he will find item B before he gets to the next store offering item A or not. If he does not find item B before he gets to the next store offering item A, the system might recommend a store in which the user can buy both items A and B together. This system behavior implied by the policy complicates the analysis of upcoming stores. The problem can be solved by making additional assumptions, for example, that the user finds no items at all during the simulation, or that he finds all other items that are in principle available along the way following the recommendations given by the simulated execution of the policy.

Overall Rating: During the decision-theoretic planning process, the system computes the expected utilities of alternative options in each state. This is a standard characteristic of algorithms like value iteration or policy iteration (cf. Section 3.1.4). The object-oriented representation that we have described in Section 3.1.5.2 allows easy access to this information. Expected utility might be the best source of information to give an overall rating of alternative navigation recommendations because it takes the whole range of information on which the decision-theoretic planning process is based on into account.

A simple approach to map expected utility to a discrete number of ratings (e.g., a five star rating or a rating with five different facial expressions of a smiley) works as follows: Compute the difference of the expected utility of the best and the worst option; divide this difference by the number of discrete grades; add the resulting numbers to the minimal expected utility (the expected utility of the worst option) to gain a number of expected utility grades; map each expected utility to the discrete grade that corresponds to the closest expected utility grade.

Difficulty of Following the Directions: A quantitative measure for the difficulty of following the upcoming directions has already been indicated in the questionnaire: the expected number of turning points for alternative navigation options. The required information can be ascertained by similar simulations as described with regard to information about stores along the way.

Best and worst cases can be determined by assuming that the user finds every desired item at the first attempt or that he does not find any item at all. The average number of turning points can be determined by repeated simulation during which the success about finding an item or not is determined according to the given probabilities.

In the decision-theoretic models described in Section 4.2.3, we have shown how the possibility that the user takes a wrong turn can be considered. The probability of taking a wrong turn can be used to derive the difficulty of following directions, in particular if the horizon for a look ahead is short. For example, comparing two alternative routes, the system could determine the total probability to take a wrong turn when trying to follow the next n navigation recommendations.

7.5 Synopsis

We have presented three user studies in this chapter. By the first two studies, we learned about how users react to a PDA-based shopping guide combining product location with navigation support based on a decision-theoretic policy. The first study, conducted in a mock-up of a shopping mall, confirmed our expectation that users would prefer our PDA-based system over a traditional paper map: A large majority of the subjects accepted the application of a decision-theoretic shopping guide and stated that they would like to use such a system in the future in situations similar to the situation of the first user study. Furthermore, we found that the subjects managed to complete their shopping task more quickly. On the basis of the feedback given by the subjects in the questionnaire used in the first user study, we improved the shopping guide with features such as overview maps and information about upcoming shops—the improved shopping guide did not feature an explanation component, yet.

The fact that we conducted the first user study in a mock-up of a shopping mall was often criticized. Therefore, we asked the subjects in a second user study to use the improved version of the shopping guide in a real shopping mall. In the improved shopping guide, we have also employed an enhanced decision-theoretic model taking a fixed deadline into account—a feature that is particularly important in an airport shopping scenario. In the second user study, the experimenter simulated the infrared localization technology that was applied in the first user study. The second user study confirmed that users accept the idea of a PDA-based decision-theoretic shopping guide also in a real shopping mall and a realistic shopping situation. Moreover, the second user study substantiated our expectation that the decision-theoretic approach would be applicable to a real world problem of reasonable size. However, the overall system can still be improved with regard to the usability requirements of everyday life application.

As an example for further improvement of the shopping guide, we explored in a third user study the possibilities of enhancing a decision-theoretic shopping guide by an explanation component. We presented the subjects a questionnaire by which we tried to elicit what kind of information users of a shopping guide might find useful as explanations for the system's decisions. Thereby, the first part of the questionnaire focused on the information content, whereas the second part of the questionnaire focused on the information representation. The subjects assessed those information that relates to primary and secondary goals in the decision-theoretic planning process as most useful: the expected time until arrival at the gate (primary goal) and information about upcoming stores (secondary goals). We have described approaches (mainly based on repeated simulations) to acquire the necessary information for providing such explanations of navigation recommendations. The implementation of these approaches goes beyond the scope of this thesis.

8 CONCLUSIONS

Decision-theoretic planning is applied in domains as diverse as the planning of business processes and robot navigation control. Our motivating example in the introduction of this thesis illustrated the usefulness of decision-theoretic planning in a situation of everyday life and demonstrated the benefit of a complex look-ahead taking the consequences of uncertain future actions into account.

8.1 Contributions

In this thesis, we showed how decision-theoretic planning can successfully be applied in the context of user-adaptive systems. The contributions of this thesis can be arranged in three groups: (1) General contributions concerning the development and application of decision-theoretic models for the planning in user-adaptive systems; (2) the analysis of usability issues and in particular the empirical validation of the decision-theoretic planning approach for user-adaptive systems; and (3) the consideration of special requirements of specific types of user-adaptive application. In the following, we will itemize the individual contributions of these three groups.

General contributions concerning the development and application of decision-theoretic models for the planning in user-adaptive systems

- *We described the general workflow for the development and application of decision-theoretic models for planning in the context of user-adaptive systems:* The workflow description provides guidelines for all steps of the development process starting from checking the applicability of decision-theoretic planning for the problem at hand to the actual application of a policy. The workflow description paves the way for the development of a tool that supports the designer of a user-adaptive system when incorporating decision-theoretic planning. In addition, we summarized several approaches to determine the parameters required for a decision-theoretic model in a structured way.

- *We extended the workflow description such that it covers the development and application of decision-theoretic models for planning complex problems involving multiple goals:* The extension of the workflow describes how a decision-theoretic model can be developed and applied for complex, decomposable problems that can be managed by goal-priorized decision-theoretic planning, a special form of hierarchical decision-theoretic planning.

- *We analyzed the pros and cons of factored and object-oriented representations:* We explained the differences of a factored representation (in particular, the SPUDD approach) and an object-oriented representation of a Markov decision process. We analyzed the pros and cons of these representations and motivated our decision to use the object-oriented state space representation for all prototypical implementations reported in this thesis. The object-oriented state space representation allows a structured implementation of Markov decision problems, that is, a direct mapping from a state-transition diagram to an object structure.

- *We exploited existing empirical data for the development of a decision-theoretic model for a planning problem in an experimental setting:* A large data base with empirical data describing the effects of different strategies of a system interacting with a number of subjects has been build on the basis of an experiment preceding the research described in this thesis. This data base contains two types of information for different instruction strategies: the probabilities that the user can execute instructions correctly and the costs for the user's actions in terms of the time needed to execute the system's instructions. We developed a decision-theoretic model that exploits this kind of information to plan the system's instruction strategy ahead and thereby optimizes the interaction of the system with a user according to the maximum expected utility for the user. It is a strength of the approach that it allows the system to trade-off competing goals: executing all instructions as fast as possible versus executing all instructions correctly.

- *We dealt with the problem of planning instruction sequences in generalizable way:* The decision-theoretic model build on the basis of the empirical data of the experiment was formulated in a way that it can be applied to problems that are similar to the problem considered in the experiment. We adapted the decision-theoretic model developed for the experimental setting and applied the adapted model to plan ahead the interaction of a system providing assistance for a user trying to operate a credit card phone. In this context, we discussed different forms of output presentation that could be used by a system to adapt to the user's situation and possible resource limitations.

- *We developed GPDTP, a goal-priorized decision-theoretic planning approach dealing with fixed deadlines:* The exploration of more complex domains, that is, domains featuring an increased number of goals, revealed that the planning based on a single exact decision-theoretic planning process is not tractable. We developed a hierarchical decision-theoretic planning approach based on goal priorization, which takes advantage of a distinctive consideration of primary, secondary, and in principle n-ary goals, as well as the existence of a fixed deadline for the overall task. Thereby, scalability is achieved by decomposing the problem such that it can be split up and solved in decision-theoretic planning processes on two (macro-level and micro-level decision-theoretic planning) or more levels.

- *We described a method to incorporate partially observable information by the interplay of a hierarchical decision-theoretic planner and a user modeling component:* Today's best known methods to solve partially observable Markov decision process exactly do not scale well. In this thesis, we explained in how far the interplay of a hierarchical decision-theoretic planner and a user modeling component can compensate for not considering partially observable aspects of a planning problem explicitely in terms of a partially observable Markov decision process.

Analysis of usability issues and empirical validation of the decision-theoretic planning approach for user-adaptive systems

- *We validated the basic decision-theoretic approach in a user study conducted in a mock-up of a shopping mall:* In a first study, conducted in a mock-up of a shopping mall, we showed that people are in principle willing to use systems like the PDA-based decision-theoretic shopping guide, and that the use of such a system is superior to a traditional navigation aid such as a paper map. Moreover, the first user study showed the principle acceptance of the hardware configuration, in particular, the infrared localization technology.

- *We improved the decision-theoretic shopping guide taking various usability issues into account:* The improved decision-theoretic shopping guide takes a fixed deadline into account and features a more transparent decision making (e.g., by overview maps indicating upcoming stores, which help the user to understand the system's recommendations and to keep the big picture). Altogether, we discussed various aspects of the decision-theoretic shopping guide in the light of the usability concepts transparency, controllability, and predictability.

- *We validated the improved decision-theoretic shopping guide in a user study conducted in a real shopping mall:* The second user study, conducted in a real shopping mall, substantiated our findings about the user acceptance of the decision-theoretic shopping guide in the first user study. In addition, it showed the feasibility of capturing the topology of a real environment (a real shopping mall) and a realistic task with the decision-theoretic model that we developed.

- *We prepared the development of an explanation component for decision-theoretic recommender systems:* It is a drawback of the decision-theoretic approach that the underlying decision criterion (expected utility) is not appropriate to generate comprehensive explanations for the recommendations given by a system that applies decision-theoretic planning. We conducted a user study to elicit what kind of information potential users of a decision-theoretic shopping guide find useful and how this information should be presented. We discussed the results of this user study and derived approaches to provide the required information for giving useful explanations.

Consideration of special requirements of specific types of application

- *We integrated a decision-theoretic planner with a user modeling component based on Bayesian networks to take potential resource limitations of the user into account:* The integration of a decision-theoretic planner and a user modeling component was exemplified with a system providing assistance for a user trying to operate a credit card phone. Thereby, the user modeling component provides estimates for two important resource limitations of the user: the user's cognitive load and his subjectively felt time pressure. This information is given in terms of probabilities, which we used to parameterize the decision-theoretic planning process.

- *We developed a decision-theoretic model for the integrated planning of navigation recommendation and information presentation:* The planning of information presentation cannot always be considered independently from the discourse for which the information presentation takes place. This dependence is in particular important for the presentation of navigation recommendations. We developed a decision-theoretic model that allows an integrated planning of navigation recommendations and their presentation.

- *We modified the decision-theoretic model for the integrated planning of navigation recommendation and information presentation such that is can deal with predefined routes:* The modification of the decision-theoretic model used for the integrated planning of navigation recommendation and information presentation allowed to incorporate and augment predefined route descriptions. Thereby, we exemplified the flexibility of the decision-theoretic planning approach, demonstrating that it is possible to separate navigation and presentation planning if this is necessary, for example, because of modularity reasons.

- *We modified the decision-theoretic model for the integrated planning of navigation recommendations and information presentation such that it can deal with fixed deadlines:* We identified two roles of time in the context of decision-theoretic planning process: time as a pure cost factor and time as a limited resource reflected by a fixed deadline. The latter role required the application of a particular kind of finite-horizon planning, in which action costs and planning stages are unified in terms of a time measure.

8.2 Limitations and Open Questions

The research concerning the application of decision-theoretic planning in the context of user-adaptive system problems left some interesting questions open and provides challenges for practical and conceptional future work.

Realization of an explanation component for decision-theoretic recommender systems: Section 7.4 provides some detailed descriptions of methods to compute the required information to be used in an explanation component for a decision-theoretic recommender system. The methods are simulative, that is, they are applied after the planning process is completed and possibly only on demand because the required simulations would typically start from the state in which an explanation is requested. It is an open question if some of the information required to give explanations can already be computed *during* the decision-theoretic planning approach, such that it is immediately available when an explanation is requested. The corresponding algorithms may work in a similar way as the iterative algorithms to compute the expected utilities of actions during the decision-theoretic planning process.

Combination of decision-theoretic planning with classical planning approaches: In particular in the context of an explanation component, the combination of decision-theoretic planning with classical planning approaches seems to represent an interesting line of research. Thereby, decision-theoretic planning can be employed for navigation and presentation tasks as in the shopping guide examples of Chapter 7, while either a decision-theoretic or a classical planning approach may provide the solution for the problem of *how to present/communicate explanations*. Existing tools like the COLLAGEN agent middleware (see Section 2.2.1.2) can possibly be exploited for this purpose. This possibility would require to make policies processable inside COLLAGEN agents.

Further exploration of possibilities to deal with complex scenarios: An extension of the decision-theoretic model that we developed for the user study described in Section 7.3 requires to maintain in each state the information about which stores have already been visited. Augmenting the given decision-theoretic model with this information makes the corresponding Markov decision problem intractable for all but small instances. Plutowski (2003) has compared four existing approaches and combined them in a heuristic approach to solve moderately sized instances from this class of problems. In a different domain, Meuleau, Dearden, and Washington (2004) conclude recently that problems with an increased number of goals can only be tackled with heuristic and approximate approaches. The exploration of such approaches to solve larger instances from the class of problems that we consider in this thesis seems to be worthwhile if corresponding systems are supposed to be put into practice.

Analysis of the applicability of DTP for further dialog situations: We successfully applied decision-theoretic planning for the dialog planning of a system assisting a user operating a technical device in Section 4.1. A general analysis of dialog situations in the context of user-adaptive systems will probably reveal further dialog situations for which decision-theoretic planning can be beneficial. Future work may include a systematic categorization of dialog types and the development of appropriate decision-theoretic models to plan these dialogs ahead. A repository of "plug & play" decision-theoretic models (e.g., a module which takes as input a set of instructions to be given or questions to be posed by a user interface, together with a specification of different forms of output presentations including costs and probabilities of the variants' outcomes and the user's reactions, respectively) for frequent dialog situations may prepare the ground for routine application of decision-theoretic planning in particular types of user-adaptive systems.

Development and testing of further application-oriented systems: There is a range of applications similar to the shopping guide examples of Chapter 7. Systems like museum, conference, exhibition, and city/tourist guides can clearly benefit from decision-theoretic planning in the same way as the shopping guide does. Also car navigation systems and railway information systems (cf. Chapter 1) have potential for a successful application of decision-theoretic planning. Eventually, web-based recommender systems, such as the Mitos online bookstore described in Section 2.2.7.1, represent an interesting domain in the context of user-adaptive systems, which we have not explored within the scope of this thesis. Such systems will presumably lead to new requirements and challenges.

A tool to support the implementation of decision-theoretic planning for user-adaptive system problems: The general framework for the development of decision-theoretic models for user-adaptive system problems can serve as a basis for the implementation of a tool to support user-adaptive system designers who want to incorporate decision-theoretic planning to enhance the adaptive capabilities of a system. The main challenge with regard to a tool for the automatic generation of a decision-theoretic model will be the development of an interpreter that builds a fully observable Markov decision process from declarative construction rules. The state space representation with object-oriented state-transition diagrams may have advantages over factored representations for this purpose.

A SPUDD Specification of the Simple Instruction Problem

This appendix provides the specification of the simple instruction problem in a format recognizable by SPUDD[1] , version 2.0. While SPUDD in versions earlier than 2.0 allows only boolean variables as state features, SPUDD 2.0 can handle multi-valued variables as state features. That is, we can use the variables for the state features of the simple instruction problem as we have introduced them in Section 3.1.5.1:

```
(variables (n_to_give two one zero)
           (n_in_bundle two one zero)
           (n_in_wm two one zero)
           (correct yes no))
```

An action is described by its effects on the state variables. We describe two actions in Figure 3.3: GIVE INSTRUCTION and WAIT FOR EXECUTION. The action GIVE INSTRUCTION can be expressed in the SPUDD representation as follows:

```
action give_instruction 1.964
  n_to_give (n_to_give (two  (0.0 1.0 0.0))
                       (one  (0.0 0.0 1.0))
                       (zero (0.0 0.0 1.0)))
  n_in_bundle (n_in_bundle (two  (1.0 0.0 0.0))
                           (one  (1.0 0.0 0.0))
                           (zero (0.0 1.0 0.0)))
  n_in_wm (n_in_wm (two  (1.0 0.0 0.0))
                   (one  (1.0 0.0 0.0))
                   (zero (0.0 1.0 0.0)))
  correct (correct (yes (1.0 0.0))
                   (no  (0.0 1.0)))
endaction
```

The first entry of the specification means: The action GIVE INSTRUCTION changes the state feature N TO GIVE from 2 to 1, from 1 to 0, and keeps it constant if it is already 0.

SPUDD 2.0 does not allow for the specification of action costs dependent on the state features. That is, we cannot express different costs of action WAIT FOR EXECUTION in different situations. Therefore we specify three versions of the action WAIT FOR EXECUTION with different costs.

[1]**S**tochastic **P**lanning **U**sing **D**ecision **D**iagrams

Each of the three versions is only applicable under certain circumstances, e.g., WAIT FOR EXECUTION FIRST OF ONE is only applicable in the situation when N IN BUNDLE equals one and N IN WM equals one. To guarantee the restricted applicability, we have to make sure that applying the action does not lead to an undesired change of the state feature values. For example, the definition of the dynamics of the CORRECT? feature of action WAIT FOR EXECUTION FIRST OF ONE specifies that the CORRECT? feature changes to value 'no' every time it is applied in a state in which N IN BUNDLE and N IN WM do not both equal one. The restriction of the applicability is rather tricky, complex, and intransparent in the SPUDD problem representation language.

```
action wait_for_execution_first_of_one 1.814
  n_to_give (n_to_give (two  (1.0 0.0 0.0))
                       (one  (0.0 1.0 0.0))
                       (zero (0.0 0.0 1.0)))
  n_in_bundle (n_in_bundle (two  (1.0 0.0 0.0))
                           (one  (n_in_wm (two  (1.0 0.0 0.0))
                                          (one  (0.0 0.0 1.0))
                                          (zero (1.0 0.0 0.0))))
                           (zero (1.0 0.0 0.0)))
  n_in_wm (n_in_wm (two  (1.0 0.0 0.0))
                   (one  (n_in_bundle (two  (1.0 0.0 0.0))
                                      (one  (0.0 0.0 1.0))
                                      (zero (1.0 0.0 0.0))))
                   (zero (1.0 0.0 0.0)))
  correct (correct (yes (n_in_bundle (two  (0.0 1.0))
                                     (one  (n_in_wm (two  (0.0 1.0))
                                                    (one  (0.996 0.004))
                                                    (zero (0.0 1.0))))
                                     (zero (0.0 1.0))))
                   (no  (0.0 1.0)))
endaction
```

```
action wait_for_execution_first_of_two 1.518
  n_to_give (n_to_give (two  (1.0 0.0 0.0))
                       (one  (0.0 1.0 0.0))
                       (zero (0.0 0.0 1.0)))
  n_in_bundle (1.0 0.0 0.0)
  n_in_wm (n_in_wm (two  (n_in_bundle (two  (0.0 1.0 0.0))
                                      (one  (1.0 0.0 0.0))
                                      (zero (1.0 0.0 0.0))))
                   (one  (1.0 0.0 0.0))
                   (zero (1.0 0.0 0.0)))
  correct (correct (yes (n_in_bundle (two  (n_in_wm (two  (0.993 0.007))
                                                    (one  (0.0 1.0))
                                                    (zero (0.0 1.0))))
                                     (one  (0.0 1.0))
                                     (zero (0.0 1.0))))
                   (no  (0.0 1.0)))
endaction
```

```
action wait_for_execution_second_of_two 1.174
  n_to_give (n_to_give (two  (1.0 0.0 0.0))
                       (one  (0.0 1.0 0.0))
                       (zero (0.0 0.0 1.0)))
  n_in_bundle (n_in_bundle (two  (n_in_wm (two  (1.0 0.0 0.0))
                                          (one  (0.0 0.0 1.0))
                                          (zero (1.0 0.0 0.0))))
                           (one  (1.0 0.0 0.0))
                           (zero (1.0 0.0 0.0)))
  n_in_wm (n_in_wm (two  (1.0 0.0 0.0))
                   (one  (n_in_bundle (two  (0.0 0.0 1.0))
                                      (one  (1.0 0.0 0.0))
                                      (zero (1.0 0.0 0.0))))
                   (zero (1.0 0.0 0.0)))
  correct (correct (yes (n_in_bundle (two  (n_in_wm (two  (0.0 1.0))
                                                    (one  (0.971 0.029))
                                                    (zero (0.0 1.0))))
                                     (one  (0.0 1.0))
                                     (zero (0.0 1.0))))
                   (no  (0.0 1.0)))
endaction
```

Finally, we have to specify the reward function, the discount factor and the tolerance factor. The discount factor is used as we have explained in Section 3.1.4.2. The tolerance factor parameterizes SPUDD's stopping criterion.

```
reward
  (correct (yes (n_to_give
                   (two  (0.0))
                   (one  (0.0))
                   (zero (n_in_wm
                            (two  (0.0))
                            (one  (0.0))
                            (zero (n_in_bundle
                                     (two  (0.0))
                                     (one  (0.0))
                                     (zero (10.0))))))))
           (no (0.0)))

discount 0.99
tolerance 0.01
```

The example problem as it is specified in this appendix can be solved using interactive SPUDD at `http://www.cs.ubc.ca/spider/staubin/Spudd/form.html`. To compute the policy for the case where executing all instructions correctly is rewarded less, the reward function has to be modified in the canonical way.

B COMPUTATION TRACE EXAMPLE: PHONE ASSISTANCE

This appendix lists a part of the computation trace for a small instance of the planning problem for assisting a user to operate a credit card phone. We consider a sequence of 3 instructions with 1 repetition allowed. The user is not distracted. The probabilities of incorrect execution of an instruction are 0.01, 0.04, and 0.14 dependent on how long the user had to memorize an instruction before executing it. In a later version, we have refined the modeling (cf. Figure 4.2) by taking the exact position of an instruction within a bundle into account. The reward for finishing without mistake is 25, the reward for finishing with mistake 5. We assume that the duration of giving an instruction is 0.5sec, the duration of executing an instruction is 0.5sec, and the duration of confirming the execution of an instruction is 0.5sec. We present the results of the first 10 iterations of value iteration. Each pair of lines contains the following information:

- **First row of each pair**: (1) the ID of the state object, (2) the number of instructions to be given, (3) the number of instructions to be executed, (4) the information if all executed instructions were executed correctly, where + corresponds to *true* and – corresponds to *false*, and (5) the content of the user's working memory—the number in brackets indicates how long the user had to memorize an instruction in the given state, before he can execute it;
- **Second row of each pair**:(1) the utility of the state, which equals the expected utility of the best action for the given iteration, (2) the optimal action for the given iteration (g: give, e: execute, r:repeat), and (3) the expected utilities of the available actions (give, execute, repeat) in the given state for the given iteration—nodef means that the corresponding action is not available.

```
------------------------------------- Iteration: 1 ----------------------------
1 3 3 + nil
utility: ~1.0  opt. action: g  <--- give: ~1.0  exe: nodef  repeat: nodef
2 2 3 + [i(0)]
utility: ~0.5  opt. action: g  <--- give: ~0.5  exe: ~1.0  repeat: nodef
3 1 3 + [i(1) i(0)]
utility: ~0.5  opt. action: g  <--- give: ~0.5  exe: ~1.0  repeat: nodef
4 0 3 + [i(2) i(1) i(0)]
utility: ~1.0  opt. action: e  <--- give: nodef  exe: ~1.0  repeat: nodef
5 0 2 + [i(2) i(1)]
utility: ~1.0  opt. action: e  <--- give: nodef  exe: ~1.0  repeat: nodef
6 0 1 + [i(2)]
utility: 23.0  opt. action: e  <--- give: nodef  exe: 23.0  repeat: nodef
8 0 0 - nil
utility: ~1.0  opt. action: r  <--- give: nodef  exe: nodef  repeat: ~1.0
```

```
9 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
12 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
14 0 2 - [i(2) i(1)]
utility: ˜0.5  opt. action: e  <--- give: nodef  exe: ˜0.5  repeat: nodef
15 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
17 1 2 + [i(1)]
utility: ˜1.0  opt. action: e  <--- give: nodef  exe: ˜1.0  repeat: nodef
18 1 1 + nil
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: nodef  repeat: nodef
19 0 1 + [i(0)]
utility: 23.75  opt. action: e  <--- give: nodef  exe: 23.75  repeat: nodef
20 1 1 - nil
utility: ˜1.0  opt. action: r  <--- give: nodef  exe: nodef  repeat: ˜1.0
21 1 2 + [i(0)]
utility: ˜1.0  opt. action: e  <--- give: nodef  exe: ˜1.0  repeat: nodef
22 1 1 + nil
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: nodef  repeat: nodef
23 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
24 1 1 - nil
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: nodef  repeat: nodef
25 0 1 - [i(0)]
utility: ˜0.45  opt. action: e  <--- give: nodef  exe: ˜0.45  repeat: nodef
26 1 2 - [i(1)]
utility: ˜0.5  opt. action: e  <--- give: nodef  exe: ˜0.5  repeat: nodef
27 1 1 - nil
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: nodef  repeat: nodef
28 0 1 - [i(0)]
utility: ˜0.5  opt. action: e  <--- give: nodef  exe: ˜0.5  repeat: nodef
29 2 2 + nil
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: nodef  repeat: nodef
30 1 2 + [i(0)]
utility: ˜0.5  opt. action: g  <--- give: ˜0.5  exe: ˜1.0  repeat: nodef
31 0 2 + [i(1) i(0)]
utility: ˜1.0  opt. action: e  <--- give: nodef  exe: ˜1.0  repeat: nodef
32 0 1 + [i(1)]
utility: 23.5  opt. action: e  <--- give: nodef  exe: 23.5  repeat: nodef
33 0 1 - [i(1)]
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
34 2 2 - nil
utility: ˜1.0  opt. action: r  <--- give: nodef  exe: nodef  repeat: ˜1.0
35 2 3 + [i(0)]
utility: ˜1.0  opt. action: e  <--- give: nodef  exe: ˜1.0  repeat: nodef
36 2 2 + nil
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: nodef  repeat: nodef
37 1 2 + [i(0)]
utility: ˜0.5  opt. action: g  <--- give: ˜0.5  exe: ˜1.0  repeat: nodef
38 0 2 + [i(1) i(0)]
utility: ˜1.0  opt. action: e  <--- give: nodef  exe: ˜1.0  repeat: nodef
39 0 1 + [i(1)]
utility: 23.6  opt. action: e  <--- give: nodef  exe: 23.6  repeat: nodef
40 0 1 - [i(1)]
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
42 2 2 - nil
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: nodef  repeat: nodef
43 1 2 - [i(0)]
utility: ˜0.5  opt. action: e  <--- give: ˜0.5  exe: ˜0.5  repeat: nodef
44 0 2 - [i(1) i(0)]
utility: ˜0.5  opt. action: e  <--- give: nodef  exe: ˜0.5  repeat: nodef
-------------------------------Iteration: 2  ---------------------------------
1 3 3 + nil
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: nodef  repeat: nodef
```

```
2 2 3 + [i(0)]
utility: ˜1.0  opt. action: g  <--- give: ˜1.0  exe: ˜2.0  repeat: nodef
3 1 3 + [i(1) i(0)]
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: ˜1.99  repeat: nodef
4 0 3 + [i(2) i(1) i(0)]
utility: ˜1.98  opt. action: e  <--- give: nodef  exe: ˜1.98  repeat: nodef
5 0 2 + [i(2) i(1)]
utility: 21.068  opt. action: e  <--- give: nodef  exe: 21.068  repeat: nodef
6 0 1 + [i(2)]
utility: 22.96  opt. action: e  <--- give: nodef  exe: 22.96  repeat: nodef
8 0 0 - nil
utility: 22.8  opt. action: r  <--- give: nodef  exe: nodef  repeat: 22.8
9 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
12 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
14 0 2 - [i(2) i(1)]
utility: ˜0.512  opt. action: e  <--- give: nodef  exe: ˜0.512  repeat: nodef
15 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
17 1 2 + [i(1)]
utility: ˜2.0  opt. action: e  <--- give: nodef  exe: ˜2.0  repeat: nodef
18 1 1 + nil
utility: 22.75  opt. action: g  <--- give: 22.75  exe: nodef  repeat: nodef
19 0 1 + [i(0)]
utility: 23.74  opt. action: e  <--- give: nodef  exe: 23.74  repeat: nodef
20 1 1 - nil
utility: ˜2.0  opt. action: r  <--- give: nodef  exe: nodef  repeat: ˜2.0
21 1 2 + [i(0)]
utility: ˜2.0  opt. action: e  <--- give: nodef  exe: ˜2.0  repeat: nodef
22 1 1 + nil
utility: 22.8  opt. action: g  <--- give: 22.8  exe: nodef  repeat: nodef
23 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
24 1 1 - nil
utility: ˜1.45  opt. action: g  <--- give: ˜1.45  exe: nodef  repeat: nodef
25 0 1 - [i(0)]
utility: ˜0.45  opt. action: e  <--- give: nodef  exe: ˜0.45  repeat: nodef
26 1 2 - [i(1)]
utility: ˜0.52  opt. action: e  <--- give: nodef  exe: ˜0.52  repeat: nodef
27 1 1 - nil
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: nodef  repeat: nodef
28 0 1 - [i(0)]
utility: ˜0.51  opt. action: e  <--- give: nodef  exe: ˜0.51  repeat: nodef
29 2 2 + nil
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: nodef  repeat: nodef
30 1 2 + [i(0)]
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: ˜2.0  repeat: nodef
31 0 2 + [i(1) i(0)]
utility: 22.022  opt. action: e  <--- give: nodef  exe: 22.022  repeat: nodef
32 0 1 + [i(1)]
utility: 23.48  opt. action: e  <--- give: nodef  exe: 23.48  repeat: nodef
33 0 1 - [i(1)]
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
34 2 2 - nil
utility: ˜2.0  opt. action: r  <--- give: nodef  exe: nodef  repeat: ˜2.0
35 2 3 + [i(0)]
utility: ˜2.0  opt. action: e  <--- give: nodef  exe: ˜2.0  repeat: nodef
36 2 2 + nil
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: nodef  repeat: nodef
37 1 2 + [i(0)]
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: ˜2.0  repeat: nodef
38 0 2 + [i(1) i(0)]
utility: 22.12  opt. action: e  <--- give: nodef  exe: 22.12  repeat: nodef
39 0 1 + [i(1)]
```

```
utility: 23.6  opt. action: e  <--- give: nodef  exe: 23.6  repeat: nodef
40 0 1 - [i(1)]
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
42 2 2 - nil
utility: ˜1.5  opt. action: g  <--- give: ˜1.5  exe: nodef  repeat: nodef
43 1 2 - [i(0)]
utility: ˜0.51  opt. action: e  <--- give: ˜1.0  exe: ˜0.51  repeat: nodef
44 0 2 - [i(1) i(0)]
utility: ˜0.508  opt. action: e  <--- give: nodef  exe: ˜0.508  repeat: nodef
--------------------------------Iteration: 3  ----------------------------------
1 3 3 + nil
utility: ˜2.0  opt. action: g  <--- give: ˜2.0  exe: nodef  repeat: nodef
2 2 3 + [i(0)]
utility: ˜2.0  opt. action: g  <--- give: ˜2.0  exe: ˜2.505  repeat: nodef
3 1 3 + [i(1) i(0)]
utility: ˜2.48  opt. action: g  <--- give: ˜2.48  exe: ˜2.9704  repeat: nodef
4 0 3 + [i(2) i(1) i(0)]
utility: 19.2048  opt. action: e  <--- give: nodef  exe: 19.2048  repeat: nodef
5 0 2 + [i(2) i(1)]
utility: 21.0296  opt. action: e  <--- give: nodef  exe: 21.0296  repeat: nodef
6 0 1 + [i(2)]
utility: 23.912  opt. action: e  <--- give: nodef  exe: 23.912  repeat: nodef
8 0 0 - nil
utility: 22.8  opt. action: r  <--- give: nodef  exe: nodef  repeat: 22.8
9 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
12 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
14 0 2 - [i(2) i(1)]
utility: ˜0.512  opt. action: e  <--- give: nodef  exe: ˜0.512  repeat: nodef
15 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
17 1 2 + [i(1)]
utility: 21.255  opt. action: e  <--- give: nodef  exe: 21.255  repeat: nodef
18 1 1 + nil
utility: 22.74  opt. action: g  <--- give: 22.74  exe: nodef  repeat: nodef
19 0 1 + [i(0)]
utility: 23.978  opt. action: e  <--- give: nodef  exe: 23.978  repeat: nodef
20 1 1 - nil
utility: ˜3.0  opt. action: r  <--- give: nodef  exe: nodef  repeat: ˜3.0
21 1 2 + [i(0)]
utility: 21.5575  opt. action: e  <--- give: nodef  exe: 21.5575  repeat: nodef
22 1 1 + nil
utility: 22.8  opt. action: g  <--- give: 22.8  exe: nodef  repeat: nodef
23 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
24 1 1 - nil
utility: ˜1.45  opt. action: g  <--- give: ˜1.45  exe: nodef  repeat: nodef
25 0 1 - [i(0)]
utility: ˜0.45  opt. action: e  <--- give: nodef  exe: ˜0.45  repeat: nodef
26 1 2 - [i(1)]
utility: ˜0.53  opt. action: e  <--- give: nodef  exe: ˜0.53  repeat: nodef
27 1 1 - nil
utility: ˜1.51  opt. action: g  <--- give: ˜1.51  exe: nodef  repeat: nodef
28 0 1 - [i(0)]
utility: ˜0.272  opt. action: e  <--- give: nodef  exe: ˜0.272  repeat: nodef
29 2 2 + nil
utility: ˜2.5  opt. action: g  <--- give: ˜2.5  exe: nodef  repeat: nodef
30 1 2 + [i(0)]
utility: 21.522  opt. action: g  <--- give: 21.522  exe: 21.5025  repeat: nodef
31 0 2 + [i(1) i(0)]
utility: 22.0024  opt. action: e  <--- give: nodef  exe: 22.0024  repeat: nodef
32 0 1 + [i(1)]
utility: 23.956  opt. action: e  <--- give: nodef  exe: 23.956  repeat: nodef
33 0 1 - [i(1)]
```

```
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
34 2 2 - nil
utility: ˜3.0  opt. action: r  <--- give: nodef  exe: nodef  repeat: ˜3.0
35 2 3 + [i(0)]
utility: ˜2.5  opt. action: e  <--- give: nodef  exe: ˜2.5  repeat: nodef
36 2 2 + nil
utility: ˜2.5  opt. action: g  <--- give: ˜2.5  exe: nodef  repeat: nodef
37 1 2 + [i(0)]
utility: 21.62  opt. action: g  <--- give: 21.62  exe: 21.5575  repeat: nodef
38 0 2 + [i(1) i(0)]
utility: 22.12  opt. action: e  <--- give: nodef  exe: 22.12  repeat: nodef
39 0 1 + [i(1)]
utility: 23.6  opt. action: e  <--- give: nodef  exe: 23.6  repeat: nodef
40 0 1 - [i(1)]
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
42 2 2 - nil
utility: ˜1.51  opt. action: g  <--- give: ˜1.51  exe: nodef  repeat: nodef
43 1 2 - [i(0)]
utility: ˜0.5145  opt. action: e  <--- give: ˜1.008  exe: ˜0.5145  repeat: nodef
44 0 2 - [i(1) i(0)]
utility: ˜0.508  opt. action: e  <--- give: nodef  exe: ˜0.508  repeat: nodef

...

--------------------------------Iteration: 10  ---------------------------------
1 3 3 + nil
utility: 18.9805  opt. action: g  <--- give: 18.9805  exe: nodef  repeat: nodef
2 2 3 + [i(0)]
utility: 19.9805  opt. action: g  <--- give: 19.9805  exe: 19.9432  repeat: nodef
3 1 3 + [i(1) i(0)]
utility: 20.4805  opt. action: e  <--- give: 19.5453  exe: 20.4805  repeat: nodef
4 0 3 + [i(2) i(1) i(0)]
utility: 20.0453  opt. action: e  <--- give: nodef  exe: 20.0453  repeat: nodef
5 0 2 + [i(2) i(1)]
utility: 21.9435  opt. action: e  <--- give: nodef  exe: 21.9435  repeat: nodef
6 0 1 + [i(2)]
utility: 23.912  opt. action: e  <--- give: nodef  exe: 23.912  repeat: nodef
8 0 0 - nil
utility: 22.8  opt. action: r  <--- give: nodef  exe: nodef  repeat: 22.8
9 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
12 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
14 0 2 - [i(2) i(1)]
utility: ˜0.512  opt. action: e  <--- give: nodef  exe: ˜0.512  repeat: nodef
15 0 1 - [i(2)]
utility: ˜0.3  opt. action: e  <--- give: nodef  exe: ˜0.3  repeat: nodef
17 1 2 + [i(1)]
utility: 21.9296  opt. action: e  <--- give: nodef  exe: 21.9296  repeat: nodef
18 1 1 + nil
utility: 22.978  opt. action: g  <--- give: 22.978  exe: nodef  repeat: nodef
19 0 1 + [i(0)]
utility: 23.978  opt. action: e  <--- give: nodef  exe: 23.978  repeat: nodef
20 1 1 - nil
utility: 20.5575  opt. action: r  <--- give: nodef  exe: nodef  repeat: 20.5575
21 1 2 + [i(0)]
utility: 21.5575  opt. action: e  <--- give: nodef  exe: 21.5575  repeat: nodef
22 1 1 + nil
utility: 22.8  opt. action: g  <--- give: 22.8  exe: nodef  repeat: nodef
23 0 1 + [i(0)]
utility: 23.8  opt. action: e  <--- give: nodef  exe: 23.8  repeat: nodef
24 1 1 - nil
utility: ˜1.45  opt. action: g  <--- give: ˜1.45  exe: nodef  repeat: nodef
25 0 1 - [i(0)]
utility: ˜0.45  opt. action: e  <--- give: nodef  exe: ˜0.45  repeat: nodef
```

```
26 1 2 - [i(1)]
utility: ˜0.52544  opt. action: e  <--- give: nodef  exe: ˜0.52544  repeat: nodef
27 1 1 - nil
utility: ˜1.272  opt. action: g  <--- give: ˜1.272  exe: nodef  repeat: nodef
28 0 1 - [i(0)]
utility: ˜0.272  opt. action: e  <--- give: nodef  exe: ˜0.272  repeat: nodef
29 2 2 + nil
utility: 20.9689  opt. action: g  <--- give: 20.9689  exe: nodef  repeat: nodef
30 1 2 + [i(0)]
utility: 21.9689  opt. action: g  <--- give: 21.9689  exe: 21.9538  repeat: nodef
31 0 2 + [i(1) i(0)]
utility: 22.4689  opt. action: e  <--- give: nodef  exe: 22.4689  repeat: nodef
32 0 1 + [i(1)]
utility: 23.956  opt. action: e  <--- give: nodef  exe: 23.956  repeat: nodef
33 0 1 - [i(1)]
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
34 2 2 - nil
utility: 18.3987  opt. action: r  <--- give: nodef  exe: nodef  repeat: 18.3987
35 2 3 + [i(0)]
utility: 19.3987  opt. action: e  <--- give: nodef  exe: 19.3987  repeat: nodef
36 2 2 + nil
utility: 20.62  opt. action: g  <--- give: 20.62  exe: nodef  repeat: nodef
37 1 2 + [i(0)]
utility: 21.62  opt. action: g  <--- give: 21.62  exe: 21.5575  repeat: nodef
38 0 2 + [i(1) i(0)]
utility: 22.12  opt. action: e  <--- give: nodef  exe: 22.12  repeat: nodef
39 0 1 + [i(1)]
utility: 23.6  opt. action: e  <--- give: nodef  exe: 23.6  repeat: nodef
40 0 1 - [i(1)]
utility: ˜0.4  opt. action: e  <--- give: nodef  exe: ˜0.4  repeat: nodef
42 2 2 - nil
utility: ˜1.5145  opt. action: g  <--- give: ˜1.5145  exe: nodef  repeat: nodef
43 1 2 - [i(0)]
utility: ˜0.5145  opt. action: e  <--- give: ˜1.008  exe: ˜0.5145  repeat: nodef
44 0 2 - [i(1) i(0)]
utility: ˜0.508  opt. action: e  <--- give: nodef  exe: ˜0.508  repeat: nodef
```

C Initialization, Interaction Schemes, Interface Specifications

This appendix describes the processes while booting the READY prototype. Moreover, we give an example for an interaction scheme and how it is processed, and we specify the interfaces for the READY inter-modular communication.

Initialization of the READY Prototype The READY prototype is started via a so called *distribution server* that allows to distribute the components of the READY system over different physical machines. The distribution server starts the *user interface* and the *interaction manager*. The interaction manager (1) loads the *interaction schemes*, which are specified in a separate file, (2) opens a new shell and starts the user modeling component, (3) connects to the user modeling component, (4) connects to the user interface via the distribution server, and (5) initializes the display with the current information given by the user model—estimations of the user's cognitive load and subjectively felt time pressure. After these initial arrangements are completed, the first interaction scheme can be processed.

Interaction Scheme Example The interaction scheme *assist_call_with_credit_card* specifies the raw dialog structure that the system will use to assist the user with operating a credit card phone. The planning will be parameterized with the up-to-date information from the user modeling component at the time it is requested. In the specification, *degreesOfDifficulty* describes the degrees of difficulty for each instruction of a set of instructions. The coarse grained scale ranges from 1 (easy), 2 (moderate), to 3 (difficult). The degree of difficulty together with the basic probabilities for not executing an instruction correctly according to its position in a bundle (*errorProbsNoDist* and *errorProbsDist* for situations without and with distraction, respectively) determine the probabilities used in the decision-theoretic planning process. The field *costsOfInstructionExe* specifies the basic costs for executing each of the individual instructions. Together with the cost factors for executing an instruction according to its position in a bundle (*exeCostsNoDist* and *exeCostsDist* for situations without and with distraction, respectively), they determine the costs used in the decision-theoretic planning process. We have translated the interaction scheme originally specified in Oz syntax one-to-one into a pseudo code syntax with slightly improved legibility.

```
assist_call_with_credit_card:
   [dtp(plan(dialog:
             [instructions:3 repetitions:3
```

```
            instructionList:["Keep your credit card ready!"
                             "Lift handset!" "Dial 0!"]
            degreesOfDifficulty:[2 1 1]
            costsOfInstructionExe:[10 4 4]
            errorProbsNoDist:[31#0.00078 32#0.02521 33#0.00840
                              21#0.00078 22#0.00078
                              11#0.00078]
            errorProbsDist:[31#0.04201 32#0.05882 33#0.06722
                            21#0.00719 22#0.02877
                            11#0.00392]
            exeCostsNoDist:[31#1.179 32#0.949 33#0.982
                            21#1.086 22#0.872
                            11#0.912]
            exeCostsDist:[31#1.530 32#1.353 33#1.133
                          21#1.518 22#1.174
                          11#0.999]
            okay:["<Okay, I’ve done that.>"]
            nokay:["<Could you please repeat?>’’]]))
dtp(plan(dialog:
            [instructions:2 repetitions:2
             instructionList:
                ["After the tone, dial 9!"
                 "After the tone, enter the credit card number!"]
             degreesOfDifficulty:[1 3]
             costsOfInstructionExe:[5 30]
             errorProbsNoDist:[21#0.00078 22#0.00078 11#0.00078]
             errorProbsDist:[21#0.00719 22#0.02877 11#0.00392]
             exeCostsNoDist:[21#1.086 22#0.872
                             11#0.912]
             exeCostsDist:[21#1.518 22#1.174
                           11#0.999]
             okay:["<Okay, I’ve done that.>"]
             nokay:["<Could you please repeat?>"]]))
dtp(plan(dialog:
            [instructions:3 repetitions:3
             instructionList:
                ["Enter two digits for month of expiration date!"
                 "Enter two digits for year of expiration date!"
                 "After the tone, dial the desired number!"]
             degreesOfDifficulty:[2 2 3]
             costsOfInstructionExe:[8 8 20]
             errorProbsNoDist:[31#0.00078 32#0.02521 33#0.00840
                               21#0.00078 22#0.00078
                               11#0.00078]
             errorProbsDist:[31#0.04201 32#0.05882 33#0.06722
                             21#0.00719 22#0.02877
                             11#0.00392]
             exeCostsNoDist:[31#1.179 32#0.949 33#0.982
                             21#1.086 22#0.872
                             11#0.912]
             exeCostsDist:[31#1.530 32#1.353 33#1.133
                           21#1.518 22#1.174
                           11#0.999]
             okay:["<Okay, I’ve done that.>"]
             nokay:["<Could you please repeat?>"]]))
im(next_scheme:toplevelScreen)]
```

Interface Specifications The interaction manager has interfaces to the graphical user interface, the user modeling component, and the decision-theoretic planner. The first two interfaces are realized by socket communication, the latter works by message passing—interaction manager and decision-theoretic planner run in the same process.

The interaction manager can send three types of information to the user interface: (1) updates of the user's cognitive load and subjectively felt time pressure; these are given in percent, (2) user input options, and (3) system output. The information about user input options is split up into the content of the user input and symptoms that can occur during the user input. Here is an example for a message specifying user input options:

```
ui(sendMessage:
       "3#102#"#
       "map;gate%vip lounge%telephone%rest room;"#
       "/home/ready2/prototyp/UI/pics/toplevelMapNew.gif#"#
       "<I'd like to go there...>;"#
       "<What can I do here..?>;"#
       "<I'd like to use this...>\n")
```

The first two numbers specify the modality combination. In the example, the user interface will provide for multi-modal (speech and gesture) input. The next row specifies that the user can point to the gate, the vip lounge, the telephone, or the rest room on a map stored at the location given in the fourth row. Then follows a sequence of three possible speech utterances. All messages used for socket communication are declared as strings and have to be parsed by the recipient components. Here is another example, specifying system output:

```
ui(sendMessage:
         "13#401#"#
         "/home/ready2/prototyp/UI/pics/pdaWithArrow.gif#"#
         "Bitte folgen Sie den Pfeilen!\n")
```

After each interaction of the user with the system, the interaction manager receives information about the user input. The received string starts with a flag for the type of user input (speech, gesture, both speech and gesture), followed by the content of the user's input and a list of symptoms that occurred during the input (speech symptoms and motor symptoms, respectively). The messages from the user interface are parsed and processed. The information about speech and motor symptoms is passed to the user modeling component, which returns updated information about the user's cognitive load and subjectively felt time pressure. The information about the updated user model is passed to the user interface, and the next step of the interaction scheme can be processed.

D Computation Trace Example: Navigation Domain

This appendix lists a part of the computation trace for an instance of planning problem for providing navigation recommendations in a small airport building. We first specify the topology represented in Figure 4.8. The specification consists of the location ID, the coordinates of the location (used for visualization of the policies), and a list of adjacent locations. For each adjacent location, it is specified in which direction from the current location the individual adjacent location is situated, what the ID of the adjacent location is, how far it is away (in terms of the average time required to get there from the location at hand), and a weight reflecting the probability of finding a gift along the way to the particular adjacent location, which can be interpreted as the number of stores along the way from the location at hand to the adjacent location.

For the example trace, the costs to get from one location to another is 0.8 time units times the distance from the location at hand to the adjacent location if recommended with a simple instruction (we call this quick mode in the trace) and 1.0 time units times the distance from the location at hand to the adjacent location if recommended with a more complex instruction giving additional information about shops along the way (we call this info mode in the trace). Analogously, the probability to find a gift along the way from one location to another is only 0.04 times the weight for the path to the adjacent location if recommended in quick mode and 0.1 times the weight for the path to the adjacent location if recommended in info mode. The cost for actually buying a gift is assumed to be 10.0 time units. For the given trace, we assume that the user is absolutely reliable when following navigation instructions, that is, the probability that the user takes a wrong turn is set to zero.

We present the results of the first 5 iterations and the 50th iteration of value iteration. Each pair of lines contains the following information:

- **First row of each pair:** (1) The ID of the state object, (2) the information if a gift has already been bought, (3) the utility of the state, which equals the expected utility of the best action for the given iteration, and (4) the optimal action for the given iteration (mode: quick or info; direction: n, e, s, w);
- **Second row of each pair:** the expected utilities of all available actions (q stand for quick mode, i for info mode).

```
TopologyDesc = [[id:1 loc:l(1 1) links:[link(o:n id:2 d:4 w:3)
                                   link(o:e id:13 d:6 w:0)]]
                [id:2 loc:l(1 5) links:[link(o:n id:3 d:4 w:2)
                                   link(o:e id:5 d:2 w:1)
```

```
                                link(o:s id:1 d:4 w:3)]]
             [id:3 loc:l(1 9) links:[link(o:e id:9 d:3 w:5)
                                link(o:s id:2 d:4 w:2)]]
             [id:4 loc:l(3 3) links:[link(o:n id:5 d:20 w:2)]]  %% traffic
             [id:5 loc:l(3 5) links:[link(o:n id:6 d:20 w:0)    %% traffic
                                link(o:e id:7 d:1 w:0)
                                link(o:s id:4 d:20 w:2)    %% traffic
                                link(o:w id:2 d:2 w:1)]]
             [id:6 loc:l(3 6) links:[link(o:e id:8 d:1 w:0)
                                link(o:s id:5 d:20 w:0)]]  %% traffic
             [id:7 loc:l(4 5) links:[link(o:n id:8 d:2 w:0)
                                link(o:e id:11 d:1 w:0)
                                link(o:w id:5 d:1 w:0)]]
             [id:8 loc:l(4 6) links:[link(o:n id:9 d:2 w:2)
                                link(o:e id:12 d:1 w:0)
                                link(o:s id:7 d:2 w:0)
                                link(o:w id:6 d:1 w:0)]]
             [id:9 loc:l(4 9) links:[link(o:e id:15 d:3 w:2)
                                link(o:s id:8 d:2 w:2)
                                link(o:w id:3 d:3 w:5)]]
             [id:10 loc:l(5 3) links:[link(o:n id:11 d:20 w:2)]] %% traffic
             [id:11 loc:l(5 5) links:[link(o:n id:12 d:20 w:0)   %% traffic
                                 link(o:e id:14 d:2 w:1)
                                 link(o:s id:10 d:20 w:2)   %% traffic
                                 link(o:w id:7 d:1 w:0)]]
             [id:12 loc:l(5 6) links:[link(o:s id:11 d:20 w:0)   %% traffic
                                 link(o:w id:8 d:1 w:0)]]
             [id:13 loc:l(7 1) links:[link(o:n id:14 d:4 w:0)
                                 link(o:w id:1 d:6 w:0)]]
             [id:14 loc:l(7 5) links:[link(o:n id:15 d:4 w:2)
                                 link(o:e id:17 d:2 w:0)
                                 link(o:s id:13 d:4 w:0)
                                 link(o:w id:11 d:2 w:1)]]
             [id:15 loc:l(7 9) links:[link(o:s id:14 d:4 w:2)
                                 link(o:w id:9 d:3 w:2)]]
             [id:16 loc:l(9 1) links:nil]
             [id:17 loc:l(9 5) links:[link(o:n id:18 d:4 w:0)
                                 link(o:s id:16 d:4 w:0)
                                 link(o:w id:14 d:2 w:0)]]
             [id:18 loc:l(9 9) links:nil]]


----------- Iteration: 1 ------------
1   gift:false ˜4.4 quick(n)
[q(n) ˜4.4 i(n) ˜7.0 q(e) ˜4.8 i(e) ˜6.0]
2   gift:false ˜2.0 quick(e)
[q(n) ˜4.0 i(n) ˜6.0 q(e) ˜2.0 i(e) ˜3.0 q(s) ˜4.4 i(s) ˜7.0]
3   gift:false ˜4.0 quick(s)
[q(e) ˜4.4 i(e) ˜8.0 q(s) ˜4.0 i(s) ˜6.0]
4   gift:false ˜16.8 quick(n)
[q(n) ˜16.8 i(n) ˜22.0]
5   gift:false ˜0.8 quick(e)
[q(n) ˜16.0 i(n) ˜20.0 q(e) ˜0.8 i(e) ˜1.0 q(s) ˜16.8 i(s) ˜22.0 q(w) ˜2.0 i(w) ˜3.0]
6   gift:false ˜0.8 quick(e)
[q(e) ˜0.8 i(e) ˜1.0 q(s) ˜16.0 i(s) ˜20.0]
7   gift:false ˜0.8 quick(e)
[q(n) ˜1.6 i(n) ˜2.0 q(e) ˜0.8 i(e) ˜1.0 q(w) ˜0.8 i(w) ˜1.0]
8   gift:false ˜0.8 quick(e)
[q(n) ˜2.4 i(n) ˜4.0 q(e) ˜0.8 i(e) ˜1.0 q(s) ˜1.6 i(s) ˜2.0 q(w) ˜0.8 i(w) ˜1.0]
9   gift:false ˜2.4 quick(s)
[q(e) ˜3.2 i(e) ˜5.0 q(s) ˜2.4 i(s) ˜4.0 q(w) ˜4.4 i(w) ˜8.0]
10   gift:false ˜16.8 quick(n)
[q(n) ˜16.8 i(n) ˜22.0]
11   gift:false ˜0.8 quick(w)
[q(n) ˜16.0 i(n) ˜20.0 q(e) ˜2.0 i(e) ˜3.0 q(s) ˜16.8 i(s) ˜22.0 q(w) ˜0.8 i(w) ˜1.0]
```

```
12   gift:false ˜0.8 quick(w)
[q(s) ˜16.0 i(s) ˜20.0 q(w) ˜0.8 i(w) ˜1.0]
13   gift:false ˜3.2 quick(n)
[q(n) ˜3.2 i(n) ˜4.0 q(w) ˜4.8 i(w) ˜6.0]
14   gift:false ˜1.6 quick(e)
[q(n) ˜4.0 i(n) ˜6.0 q(e) ˜1.6 i(e) ˜2.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜2.0 i(w) ˜3.0]
15   gift:false ˜3.2 quick(w)
[q(s) ˜4.0 i(s) ˜6.0 q(w) ˜3.2 i(w) ˜5.0]
16   gift:false 0.0 none
nil
17   gift:false ˜1.6 quick(w)
[q(n) ˜3.2 i(n) ˜4.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜1.6 i(w) ˜2.0]
18   gift:false 0.0 none
nil
1   gift:true ˜3.2 quick(n)
[q(n) ˜3.2 i(n) ˜4.0 q(e) ˜4.8 i(e) ˜6.0]
2   gift:true ˜1.6 quick(e)
[q(n) ˜3.2 i(n) ˜4.0 q(e) ˜1.6 i(e) ˜2.0 q(s) ˜3.2 i(s) ˜4.0]
3   gift:true ˜2.4 quick(e)
[q(e) ˜2.4 i(e) ˜3.0 q(s) ˜3.2 i(s) ˜4.0]
4   gift:true ˜16.0 quick(n)
[q(n) ˜16.0 i(n) ˜20.0]
5   gift:true ˜0.8 quick(e)
[q(n) ˜16.0 i(n) ˜20.0 q(e) ˜0.8 i(e) ˜1.0 q(s) ˜16.0 i(s) ˜20.0 q(w) ˜1.6 i(w) ˜2.0]
6   gift:true ˜0.8 quick(e)
[q(e) ˜0.8 i(e) ˜1.0 q(s) ˜16.0 i(s) ˜20.0]
7   gift:true ˜0.8 quick(e)
[q(n) ˜1.6 i(n) ˜2.0 q(e) ˜0.8 i(e) ˜1.0 q(w) ˜0.8 i(w) ˜1.0]
8   gift:true ˜0.8 quick(e)
[q(n) ˜1.6 i(n) ˜2.0 q(e) ˜0.8 i(e) ˜1.0 q(s) ˜1.6 i(s) ˜2.0 q(w) ˜0.8 i(w) ˜1.0]
9   gift:true ˜1.6 quick(s)
[q(e) ˜2.4 i(e) ˜3.0 q(s) ˜1.6 i(s) ˜2.0 q(w) ˜2.4 i(w) ˜3.0]
10   gift:true ˜16.0 quick(n)
[q(n) ˜16.0 i(n) ˜20.0]
11   gift:true ˜0.8 quick(w)
[q(n) ˜16.0 i(n) ˜20.0 q(e) ˜1.6 i(e) ˜2.0 q(s) ˜16.0 i(s) ˜20.0 q(w) ˜0.8 i(w) ˜1.0]
12   gift:true ˜0.8 quick(w)
[q(s) ˜16.0 i(s) ˜20.0 q(w) ˜0.8 i(w) ˜1.0]
13   gift:true ˜3.2 quick(n)
[q(n) ˜3.2 i(n) ˜4.0 q(w) ˜4.8 i(w) ˜6.0]
14   gift:true ˜1.6 quick(e)
[q(n) ˜3.2 i(n) ˜4.0 q(e) ˜1.6 i(e) ˜2.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜1.6 i(w) ˜2.0]
15   gift:true ˜2.4 quick(w)
[q(s) ˜3.2 i(s) ˜4.0 q(w) ˜2.4 i(w) ˜3.0]
16   gift:true 0.0 none
nil
17   gift:true 16.8 quick(n)
[q(n) 16.8 i(n) 16.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜1.6 i(w) ˜2.0]
18   gift:true 20.0 none
nil
  ----------- Iteration: 2 ------------
1   gift:false ˜6.352 quick(n)
[q(n) ˜6.352 i(n) ˜8.88 q(e) ˜8.0 i(e) ˜9.2]
2   gift:false ˜2.8 quick(e)
[q(n) ˜7.872 i(n) ˜9.68 q(e) ˜2.8 i(e) ˜3.8 q(s) ˜8.656 i(s) ˜11.04]
3   gift:false ˜5.968 quick(s)
[q(e) ˜6.64 i(e) ˜10.0 q(s) ˜5.968 i(s) ˜7.92]
4   gift:false ˜17.6 quick(n)
[q(n) ˜17.6 i(n) ˜22.8]
5   gift:false ˜1.6 quick(e)
[q(n) ˜16.8 i(n) ˜20.8 q(e) ˜1.6 i(e) ˜1.8 q(s) ˜33.536 i(s) ˜38.64 q(w) ˜3.984
 i(w) ˜4.96]
6   gift:false ˜1.6 quick(e)
[q(e) ˜1.6 i(e) ˜1.8 q(s) ˜16.8 i(s) ˜20.8]
7   gift:false ˜1.6 quick(e)
```

```
[q(n) ˜2.4 i(n) ˜2.8 q(e) ˜1.6 i(e) ˜1.8 q(w) ˜1.6 i(w) ˜1.8]
8   gift:false ˜1.6 quick(e)
[q(n) ˜4.736 i(n) ˜6.24 q(e) ˜1.6 i(e) ˜1.8 q(s) ˜2.4 i(s) ˜2.8 q(w) ˜1.6 i(w) ˜1.8]
9   gift:false ˜3.2 quick(s)
[q(e) ˜6.336 i(e) ˜8.04 q(s) ˜3.2 i(s) ˜4.8 q(w) ˜8.08 i(w) ˜11.2]
10   gift:false ˜17.6 quick(n)
[q(n) ˜17.6 i(n) ˜22.8]
11   gift:false ˜1.6 quick(w)
[q(n) ˜16.8 i(n) ˜20.8 q(e) ˜3.6 i(e) ˜4.6 q(s) ˜33.536 i(s) ˜38.64 q(w) ˜1.6
 i(w) ˜1.8]
12   gift:false ˜1.6 quick(w)
[q(s) ˜16.8 i(s) ˜20.8 q(w) ˜1.6 i(w) ˜1.8]
13   gift:false ˜4.8 quick(n)
[q(n) ˜4.8 i(n) ˜5.6 q(w) ˜9.2 i(w) ˜10.4]
14   gift:false ˜2.8 quick(w)
[q(n) ˜7.136 i(n) ˜9.04 q(e) ˜3.2 i(e) ˜3.6 q(s) ˜6.4 i(s) ˜7.2 q(w) ˜2.8
 i(w) ˜3.8]
15   gift:false ˜5.536 quick(w)
[q(s) ˜5.6 i(s) ˜7.6 q(w) ˜5.536 i(w) ˜7.24]
16   gift:false 0.0 none
nil
17   gift:false ˜3.2 quick(n)
[q(n) ˜3.2 i(n) ˜4.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜3.2 i(w) ˜3.6]
18   gift:false 0.0 none
nil
1   gift:true ˜4.8 quick(n)
[q(n) ˜4.8 i(n) ˜5.6 q(e) ˜8.0 i(e) ˜9.2]
2   gift:true ˜2.4 quick(e)
[q(n) ˜5.6 i(n) ˜6.4 q(e) ˜2.4 i(e) ˜2.8 q(s) ˜6.4 i(s) ˜7.2]
3   gift:true ˜4.0 quick(e)
[q(e) ˜4.0 i(e) ˜4.6 q(s) ˜4.8 i(s) ˜5.6]
4   gift:true ˜16.8 quick(n)
[q(n) ˜16.8 i(n) ˜20.8]
5   gift:true ˜1.6 quick(e)
[q(n) ˜16.8 i(n) ˜20.8 q(e) ˜1.6 i(e) ˜1.8 q(s) ˜32.0 i(s) ˜36.0 q(w) ˜3.2
 i(w) ˜3.6]
6   gift:true ˜1.6 quick(e)
[q(e) ˜1.6 i(e) ˜1.8 q(s) ˜16.8 i(s) ˜20.8]
7   gift:true ˜1.6 quick(e)
[q(n) ˜2.4 i(n) ˜2.8 q(e) ˜1.6 i(e) ˜1.8 q(w) ˜1.6 i(w) ˜1.8]
8   gift:true ˜1.6 quick(e)
[q(n) ˜3.2 i(n) ˜3.6 q(e) ˜1.6 i(e) ˜1.8 q(s) ˜2.4 i(s) ˜2.8 q(w) ˜1.6 i(w) ˜1.8]
9   gift:true ˜2.4 quick(s)
[q(e) ˜4.8 i(e) ˜5.4 q(s) ˜2.4 i(s) ˜2.8 q(w) ˜4.8 i(w) ˜5.4]
10   gift:true ˜16.8 quick(n)
[q(n) ˜16.8 i(n) ˜20.8]
11   gift:true ˜1.6 quick(w)
[q(n) ˜16.8 i(n) ˜20.8 q(e) ˜3.2 i(e) ˜3.6 q(s) ˜32.0 i(s) ˜36.0 q(w) ˜1.6
 i(w) ˜1.8]
12   gift:true ˜1.6 quick(w)
[q(s) ˜16.8 i(s) ˜20.8 q(w) ˜1.6 i(w) ˜1.8]
13   gift:true ˜4.8 quick(n)
[q(n) ˜4.8 i(n) ˜5.6 q(w) ˜8.0 i(w) ˜9.2]
14   gift:true 15.2 quick(e)
[q(n) ˜5.6 i(n) ˜6.4 q(e) 15.2 i(e) 14.8 q(s) ˜6.4 i(s) ˜7.2 q(w) ˜2.4 i(w) ˜2.8]
15   gift:true ˜4.0 quick(w)
[q(s) ˜4.8 i(s) ˜5.6 q(w) ˜4.0 i(w) ˜4.6]
16   gift:true 0.0 none
nil
17   gift:true 16.8 quick(n)
[q(n) 16.8 i(n) 16.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜3.2 i(w) ˜3.6]
18   gift:true 20.0 none
nil
  ----------- Iteration: 3 ------------
1   gift:false ˜7.152 quick(n)
```

```
[q(n) ˜7.152 i(n) ˜9.68 q(e) ˜9.6 i(e) ˜10.8]
2   gift:false ˜3.6 quick(e)
[q(n) ˜9.81056 i(n) ˜11.5744 q(e) ˜3.6 i(e) ˜4.6 q(s) ˜10.5658 i(s) ˜12.8864]
3   gift:false ˜6.768 quick(s)
[q(e) ˜7.44 i(e) ˜10.8 q(s) ˜6.768 i(s) ˜8.72]
4   gift:false ˜18.4 quick(n)
[q(n) ˜18.4 i(n) ˜23.6]
5   gift:false ˜2.4 quick(e)
[q(n) ˜17.6 i(n) ˜21.6 q(e) ˜2.4 i(e) ˜2.6 q(s) ˜34.336 i(s) ˜39.44 q(w) ˜4.784
 i(w) ˜5.76]
6   gift:false ˜2.4 quick(e)
[q(e) ˜2.4 i(e) ˜2.6 q(s) ˜17.6 i(s) ˜21.6]
7   gift:false ˜2.4 quick(e)
[q(n) ˜3.2 i(n) ˜3.6 q(e) ˜2.4 i(e) ˜2.6 q(w) ˜2.4 i(w) ˜2.6]
8   gift:false ˜2.4 quick(e)
[q(n) ˜5.536 i(n) ˜7.04 q(e) ˜2.4 i(e) ˜2.6 q(s) ˜3.2 i(s) ˜3.6 q(w) ˜2.4
 i(w) ˜2.6]
9   gift:false ˜4.0 quick(s)
[q(e) ˜8.61312 i(e) ˜10.2288 q(s) ˜4.0 i(s) ˜5.6 q(w) ˜9.9744 i(w) ˜12.984]
10   gift:false ˜18.4 quick(n)
[q(n) ˜18.4 i(n) ˜23.6]
11   gift:false ˜2.4 quick(w)
[q(n) ˜17.6 i(n) ˜21.6 q(e) ˜4.08 i(e) ˜4.0 q(s) ˜34.336 i(s) ˜39.44 q(w) ˜2.4
 i(w) ˜2.6]
12   gift:false ˜2.4 quick(w)
[q(s) ˜17.6 i(s) ˜21.6 q(w) ˜2.4 i(w) ˜2.6]
13   gift:false ˜6.0 quick(n)
[q(n) ˜6.0 i(n) ˜6.8 q(w) ˜11.152 i(w) ˜12.352]
14   gift:false ˜3.6 quick(w)
[q(n) ˜9.41312 i(n) ˜11.2288 q(e) ˜4.8 i(e) ˜5.2 q(s) ˜8.0 i(s) ˜8.8 q(w) ˜3.6
 i(w) ˜4.6]
15   gift:false ˜5.2 info(s)
[q(s) ˜5.36 i(s) ˜5.2 q(w) ˜6.336 i(w) ˜8.04]
16   gift:false 0.0 none
nil
17   gift:false ˜3.2 quick(n)
[q(n) ˜3.2 i(n) ˜4.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜4.4 i(w) ˜4.8]
18   gift:false 0.0 none
nil
1   gift:true ˜5.6 quick(n)
[q(n) ˜5.6 i(n) ˜6.4 q(e) ˜9.6 i(e) ˜10.8]
2   gift:true ˜3.2 quick(e)
[q(n) ˜7.2 i(n) ˜8.0 q(e) ˜3.2 i(e) ˜3.6 q(s) ˜8.0 i(s) ˜8.8]
3   gift:true ˜4.8 quick(e)
[q(e) ˜4.8 i(e) ˜5.4 q(s) ˜5.6 i(s) ˜6.4]
4   gift:true ˜17.6 quick(n)
[q(n) ˜17.6 i(n) ˜21.6]
5   gift:true ˜2.4 quick(e)
[q(n) ˜17.6 i(n) ˜21.6 q(e) ˜2.4 i(e) ˜2.6 q(s) ˜32.8 i(s) ˜36.8 q(w) ˜4.0
 i(w) ˜4.4]
6   gift:true ˜2.4 quick(e)
[q(e) ˜2.4 i(e) ˜2.6 q(s) ˜17.6 i(s) ˜21.6]
7   gift:true ˜2.4 quick(e)
[q(n) ˜3.2 i(n) ˜3.6 q(e) ˜2.4 i(e) ˜2.6 q(w) ˜2.4 i(w) ˜2.6]
8   gift:true ˜2.4 quick(e)
[q(n) ˜4.0 i(n) ˜4.4 q(e) ˜2.4 i(e) ˜2.6 q(s) ˜3.2 i(s) ˜3.6 q(w) ˜2.4 i(w) ˜2.6]
9   gift:true ˜3.2 quick(s)
[q(e) ˜6.4 i(e) ˜7.0 q(s) ˜3.2 i(s) ˜3.6 q(w) ˜6.4 i(w) ˜7.0]
10   gift:true ˜17.6 quick(n)
[q(n) ˜17.6 i(n) ˜21.6]
11   gift:true 13.6 quick(e)
[q(n) ˜17.6 i(n) ˜21.6 q(e) 13.6 i(e) 13.2 q(s) ˜32.8 i(s) ˜36.8 q(w) ˜2.4
 i(w) ˜2.6]
12   gift:true ˜2.4 quick(w)
[q(s) ˜17.6 i(s) ˜21.6 q(w) ˜2.4 i(w) ˜2.6]
```

```
13   gift:true 12.0 quick(n)
[q(n) 12.0 i(n) 11.2 q(w) ˜9.6 i(w) ˜10.8]
14   gift:true 15.2 quick(e)
[q(n) ˜7.2 i(n) ˜8.0 q(e) 15.2 i(e) 14.8 q(s) ˜8.0 i(s) ˜8.8 q(w) ˜3.2 i(w) ˜3.6]
15   gift:true 12.0 quick(s)
[q(s) 12.0 i(s) 11.2 q(w) ˜4.8 i(w) ˜5.4]
16   gift:true 0.0 none
nil
17   gift:true 16.8 quick(n)
[q(n) 16.8 i(n) 16.0 q(s) ˜3.2 i(s) ˜4.0 q(w) 13.6 i(w) 13.2]
18   gift:true 20.0 none
nil
  ----------- Iteration: 4    ------------
1   gift:false ˜7.952 quick(n)
[q(n) ˜7.952 i(n) ˜10.48 q(e) ˜10.8 i(e) ˜12.0]
2   gift:false ˜4.4 quick(e)
[q(n) ˜10.6106 i(n) ˜12.3744 q(e) ˜4.4 i(e) ˜5.4 q(s) ˜11.3658 i(s) ˜13.6864]
3   gift:false ˜7.568 quick(s)
[q(e) ˜8.24 i(e) ˜11.6 q(s) ˜7.568 i(s) ˜9.52]
4   gift:false ˜19.2 quick(n)
[q(n) ˜19.2 i(n) ˜24.4]
5   gift:false ˜3.2 quick(e)
[q(n) ˜18.4 i(n) ˜22.4 q(e) ˜3.2 i(e) ˜3.4 q(s) ˜35.136 i(s) ˜40.24 q(w) ˜5.584
 i(w) ˜6.56]
6   gift:false ˜3.2 quick(e)
[q(e) ˜3.2 i(e) ˜3.4 q(s) ˜18.4 i(s) ˜22.4]
7   gift:false ˜3.2 quick(e)
[q(n) ˜4.0 i(n) ˜4.4 q(e) ˜3.2 i(e) ˜3.4 q(w) ˜3.2 i(w) ˜3.4]
8   gift:false ˜3.2 quick(e)
[q(n) ˜6.336 i(n) ˜7.84 q(e) ˜3.2 i(e) ˜3.4 q(s) ˜4.0 i(s) ˜4.4 q(w) ˜3.2
 i(w) ˜3.4]
9   gift:false ˜4.8 quick(s)
[q(e) ˜7.024 i(e) ˜6.76 q(s) ˜4.8 i(s) ˜6.4 q(w) ˜10.7744 i(w) ˜13.784]
10   gift:false ˜17.92 quick(n)
[q(n) ˜17.92 i(n) ˜21.2]
11   gift:false ˜3.2 quick(w)
[q(n) ˜18.4 i(n) ˜22.4 q(e) ˜4.848 i(e) ˜4.72 q(s) ˜35.136 i(s) ˜40.24 q(w) ˜3.2
 i(w) ˜3.4]
12   gift:false ˜3.2 quick(w)
[q(s) ˜18.4 i(s) ˜22.4 q(w) ˜3.2 i(w) ˜3.4]
13   gift:false ˜6.8 quick(n)
[q(n) ˜6.8 i(n) ˜7.6 q(w) ˜11.952 i(w) ˜13.152]
14   gift:false ˜3.76 quick(w)
[q(n) ˜7.824 i(n) ˜7.76 q(e) ˜4.8 i(e) ˜5.2 q(s) ˜9.2 i(s) ˜10.0 q(w) ˜3.76
 i(w) ˜3.8]
15   gift:false ˜5.84 info(s)
[q(s) ˜6.096 i(s) ˜5.84 q(w) ˜7.136 i(w) ˜8.84]
16   gift:false 0.0 none
nil
17   gift:false ˜3.2 quick(n)
[q(n) ˜3.2 i(n) ˜4.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜5.2 i(w) ˜5.6]
18   gift:false 0.0 none
nil
1   gift:true 7.2 quick(e)
[q(n) ˜6.4 i(n) ˜7.2 q(e) 7.2 i(e) 6.0]
2   gift:true ˜4.0 quick(e)
[q(n) ˜8.0 i(n) ˜8.8 q(e) ˜4.0 i(e) ˜4.4 q(s) ˜8.8 i(s) ˜9.6]
3   gift:true ˜5.6 quick(e)
[q(e) ˜5.6 i(e) ˜6.2 q(s) ˜6.4 i(s) ˜7.2]
4   gift:true ˜18.4 quick(n)
[q(n) ˜18.4 i(n) ˜22.4]
5   gift:true ˜3.2 quick(e)
[q(n) ˜18.4 i(n) ˜22.4 q(e) ˜3.2 i(e) ˜3.4 q(s) ˜33.6 i(s) ˜37.6 q(w) ˜4.8
 i(w) ˜5.2]
6   gift:true ˜3.2 quick(e)
```

```
[q(e) ˜3.2 i(e) ˜3.4 q(s) ˜18.4 i(s) ˜22.4]
7   gift:true 12.8 quick(e)
[q(n) ˜4.0 i(n) ˜4.4 q(e) 12.8 i(e) 12.6 q(w) ˜3.2 i(w) ˜3.4]
8   gift:true ˜3.2 quick(e)
[q(n) ˜4.8 i(n) ˜5.2 q(e) ˜3.2 i(e) ˜3.4 q(s) ˜4.0 i(s) ˜4.4 q(w) ˜3.2 i(w) ˜3.4]
9   gift:true 9.6 quick(e)
[q(e) 9.6 i(e) 9.0 q(s) ˜4.0 i(s) ˜4.4 q(w) ˜7.2 i(w) ˜7.8]
10   gift:true ˜2.4 quick(n)
[q(n) ˜2.4 i(n) ˜6.4]
11   gift:true 13.6 quick(e)
[q(n) ˜18.4 i(n) ˜22.4 q(e) 13.6 i(e) 13.2 q(s) ˜33.6 i(s) ˜37.6 q(w) ˜3.2
 i(w) ˜3.4]
12   gift:true ˜2.4 quick(s)
[q(s) ˜2.4 i(s) ˜6.4 q(w) ˜3.2 i(w) ˜3.4]
13   gift:true 12.0 quick(n)
[q(n) 12.0 i(n) 11.2 q(w) ˜10.4 i(w) ˜11.6]
14   gift:true 15.2 quick(e)
[q(n) 8.8 i(n) 8.0 q(e) 15.2 i(e) 14.8 q(s) 8.8 i(s) 8.0 q(w) 12.0 i(w) 11.6]
15   gift:true 12.0 quick(s)
[q(s) 12.0 i(s) 11.2 q(w) ˜5.6 i(w) ˜6.2]
16   gift:true 0.0 none
nil
17   gift:true 16.8 quick(n)
[q(n) 16.8 i(n) 16.0 q(s) ˜3.2 i(s) ˜4.0 q(w) 13.6 i(w) 13.2]
18   gift:true 20.0 none
nil


...


  ----------- Iteration: 50 ------------
1   gift:false ˜9.802 info(n)
[q(n) ˜10.5968 i(n) ˜9.802 q(e) ˜12.8 i(e) ˜14.0]
2   gift:false ˜8.46 info(e)
[q(n) ˜10.048 i(n) ˜10.32 q(e) ˜8.624 i(e) ˜8.46 q(s) ˜12.1618 i(s) ˜11.7014]
3   gift:false ˜7.2 info(e)
[q(e) ˜8.88 i(e) ˜7.2 q(s) ˜10.9512 i(s) ˜10.688]
4   gift:false ˜22.648 quick(n)
[q(n) ˜22.648 i(n) ˜25.52]
5   gift:false ˜7.4 quick(e)
[q(n) ˜25.0 i(n) ˜29.0 q(e) ˜7.4 i(e) ˜7.6 q(s) ˜37.9562 i(s) ˜40.9184
 q(w) ˜9.7056 i(w) ˜9.574]
6   gift:false ˜9.0 quick(e)
[q(e) ˜9.0 i(e) ˜9.2 q(s) ˜23.4 i(s) ˜27.4]
7   gift:false ˜6.6 quick(e)
[q(n) ˜9.8 i(n) ˜10.2 q(e) ˜6.6 i(e) ˜6.8 q(w) ˜8.2 i(w) ˜8.4]
8   gift:false ˜8.2 quick(s)
[q(n) ˜8.992 i(n) ˜8.48 q(e) ˜9.8 i(e) ˜10.0 q(s) ˜8.2 i(s) ˜8.6 q(w) ˜9.8
 i(w) ˜10.0]
9   gift:false ˜8.0 info(w)
[q(e) ˜8.496 i(e) ˜8.04 q(s) ˜9.048 i(s) ˜8.32 q(w) ˜8.72 i(w) ˜8.0]
10   gift:false ˜21.048 quick(n)
[q(n) ˜21.048 i(n) ˜23.92]
11   gift:false ˜5.8 info(e)
[q(n) ˜25.0 i(n) ˜29.0 q(e) ˜6.0 i(e) ˜5.8 q(s) ˜36.3562 i(s) ˜39.3184
 q(w) ˜7.4 i(w) ˜7.6]
12   gift:false ˜9.0 quick(w)
[q(s) ˜21.8 i(s) ˜25.8 q(w) ˜9.0 i(w) ˜9.2]
13   gift:false ˜8.0 quick(n)
[q(n) ˜8.0 i(n) ˜8.8 q(w) ˜14.602 i(w) ˜15.802]
14   gift:false ˜4.8 quick(e)
[q(n) ˜9.296 i(n) ˜9.04 q(e) ˜4.8 i(e) ˜5.2 q(s) ˜11.2 i(s) ˜12.0 q(w) ˜7.024
 i(w) ˜6.86]
15   gift:false ˜6.8 info(s)
```

```
[q(s) ˜7.2 i(s) ˜6.8 q(w) ˜9.792 i(w) ˜9.48]
16   gift:false 0.0 none
nil
17   gift:false ˜3.2 quick(n)
[q(n) ˜3.2 i(n) ˜4.0 q(s) ˜3.2 i(s) ˜4.0 q(w) ˜6.4 i(w) ˜6.8]
18   gift:false 0.0 none
nil
1   gift:true 7.2 quick(n)
[q(n) 7.2 i(n) 6.4 q(e) 7.2 i(e) 6.0]
2   gift:true 10.4 quick(e)
[q(n) 4.0 i(n) 3.2 q(e) 10.4 i(e) 10.0 q(s) 4.0 i(s) 3.2]
3   gift:true 7.2 quick(e)
[q(e) 7.2 i(e) 6.6 q(s) 7.2 i(s) 6.4]
4   gift:true ˜4.0 quick(n)
[q(n) ˜4.0 i(n) ˜8.0]
5   gift:true 12.0 quick(e)
[q(n) ˜5.6 i(n) ˜9.6 q(e) 12.0 i(e) 11.8 q(s) ˜20.0 i(s) ˜24.0 q(w) 8.8 i(w) 8.4]
6   gift:true 10.4 quick(e)
[q(e) 10.4 i(e) 10.2 q(s) ˜4.0 i(s) ˜8.0]
7   gift:true 12.8 quick(e)
[q(n) 9.6 i(n) 9.2 q(e) 12.8 i(e) 12.6 q(w) 11.2 i(w) 11.0]
8   gift:true 11.2 quick(s)
[q(n) 8.0 i(n) 7.6 q(e) 9.6 i(e) 9.4 q(s) 11.2 i(s) 10.8 q(w) 9.6 i(w) 9.4]
9   gift:true 9.6 quick(s)
[q(e) 9.6 i(e) 9.0 q(s) 9.6 i(s) 9.2 q(w) 4.8 i(w) 4.2]
10   gift:true ˜2.4 quick(n)
[q(n) ˜2.4 i(n) ˜6.4]
11   gift:true 13.6 quick(e)
[q(n) ˜5.6 i(n) ˜9.6 q(e) 13.6 i(e) 13.2 q(s) ˜18.4 i(s) ˜22.4 q(w) 12.0
 i(w) 11.8]
12   gift:true 10.4 quick(w)
[q(s) ˜2.4 i(s) ˜6.4 q(w) 10.4 i(w) 10.2]
13   gift:true 12.0 quick(n)
[q(n) 12.0 i(n) 11.2 q(w) 2.4 i(w) 1.2]
14   gift:true 15.2 quick(e)
[q(n) 8.8 i(n) 8.0 q(e) 15.2 i(e) 14.8 q(s) 8.8 i(s) 8.0 q(w) 12.0 i(w) 11.6]
15   gift:true 12.0 quick(s)
[q(s) 12.0 i(s) 11.2 q(w) 7.2 i(w) 6.6]
16   gift:true 0.0 none
nil
17   gift:true 16.8 quick(n)
[q(n) 16.8 i(n) 16.0 q(s) ˜3.2 i(s) ˜4.0 q(w) 13.6 i(w) 13.2]
18   gift:true 20.0 none
nil
```

E Plans and Policies: Sensitivity Analysis

This appendix lists a sample for the sensitivity analysis of plans and policies in the airport navigation domain. We assume a reward of 50 to arrive at the gate (Location 18) with a gift. The maximal plan length was set to 9. For each location in the airport—starting with no gift bought—we simulated the execution of the plan (assigned to this location) and of the policy 1000 times (with a randomized finding of the gift) and computed the rewards achieved on average. In the example given, the probabilities for finding a gift that we used in the computation of the plans and the policy are 25% smaller than those used in the simulation of the plans and the policy. The following listing first describes the policy with its expected utilities for the given situation and locations, then the plans with its expected utilities for the given situation and locations, and finally, the actually achieved rewards with the plans and the policy averaged over 1000 simulation runs. Thereby, the number in brackets describes the length of the used plan and the average length of the action sequence determined by the policy, respectively.

```
'building topology...'
'computing policy...'

1  noGift  info(n)  16.6879          1  gift  quick(n)  37.2

2  noGift  quick(n)  17.8683         2  gift  quick(e)  40.4

3  noGift  info(e)  20.6769          3  gift  quick(e)  37.2

4  noGift  quick(n)  2.03594         4  gift  quick(n)  26.0

5  noGift  quick(e)  17.1446         5  gift  quick(e)  42.0

6  noGift  quick(e)  18.7446         6  gift  quick(e)  40.4

7  noGift  quick(n)  17.9446         7  gift  quick(e)  42.8

8  noGift  info(n)  19.5446          8  gift  quick(s)  41.2

9  noGift  info(w)  20.1231          9  gift  quick(e)  39.6

10  noGift  quick(n)  2.13194        10  gift  quick(n)  27.6

11  noGift  quick(w)  17.1446        11  gift  quick(e)  43.6

12  noGift  quick(w)  18.7446        12  gift  quick(w)  40.4

13  noGift  quick(n)  13.3629        13  gift  quick(n)  42.0
```

```
14  noGift  info(n)  16.5629         14  gift  quick(e)  45.2

15  noGift  info(w)  18.5446         15  gift  quick(s)  42.0

16  noGift  none  0.0                16  gift  none  0.0

17  noGift  quick(w)  14.9629        17  gift  quick(n)  46.8

18  noGift  none  0.0                18  gift  none  50.0


’generating plans...’
’evaluating plans...’

1  [info(n) quick(n) info(e) info(w) info(e) info(e) quick(s) quick(e) quick(n)]
   7.11571

2  [quick(n) info(e) info(w) info(e) info(e) quick(s) quick(e) quick(n)]
   9.46543

3  [info(e) info(w) info(e) info(e) quick(s) quick(e) quick(n)]
   12.1973

4  [quick(n) info(w) info(n) info(e) info(e) info(s) quick(e) quick(n)]
   ˜10.1495

5  [quick(e) quick(n) info(n) info(w) info(e) info(e) info(s) quick(e) quick(n)]
   8.2043

6  [quick(e) info(n) info(w) info(e) info(e) info(s) quick(e) quick(n)]
   9.8043

7  [quick(n) info(n) info(w) info(e) info(e) info(s) quick(e) quick(n)]
   9.0043

8  [info(n) info(w) info(e) info(w) info(e) quick(e) quick(s) quick(e) quick(n)]
   11.0159

9  [info(w) info(e) info(w) info(e) quick(e) quick(s) quick(e) quick(n)]
   12.2069

10  [quick(n) info(e) info(n) info(w) info(e) info(s) quick(e) quick(n)]
    ˜14.9554

11  [quick(w) quick(n) info(n) info(w) info(e) info(e) info(s) quick(e) quick(n)]
    8.2043

12  [quick(w) info(n) info(w) info(e) info(e) info(s) quick(e) quick(n)]
    9.8043

13  [quick(n) info(n) info(w) info(w) info(e) info(e) info(s) quick(e) quick(n)]
    3.84365

14  [info(n) info(w) info(w) info(e) info(e) info(s) quick(e) quick(n)]
    7.04365

15  [quick(w) info(w) info(e) info(w) info(e) quick(e) quick(s) quick(e) quick(n)]
    10.1305

17  [quick(w) info(n) info(w) info(w) info(e) info(e) info(s) quick(e) quick(n)]
    5.44365
```

```
’simulating plans and policies...’
’average rewards...’

1  plan( 9 ): 10.6  policy( 7.802 ): 19.3698

2  plan( 8 ): 12.64  policy( 7.194 ): 20.0526

3  plan( 7 ): 16.12  policy( 6.334 ): 22.7936

4  plan( 8 ): ˜6.12  policy( 9.814 ): 4.1376

5  plan( 9 ): 12.8  policy( 9.098 ): 19.2444

6  plan( 8 ): 14.52  policy( 8.088 ): 20.866

7  plan( 8 ): 13.04  policy( 8.154 ): 19.856

8  plan( 9 ): 13.6  policy( 7.082 ): 21.6726

9  plan( 8 ): 15.4  policy( 6.724 ): 21.8372

10  plan( 8 ): ˜10.52  policy( 9.586 ): 4.4066

11  plan( 9 ): 12.72  policy( 9.054 ): 19.341

12  plan( 8 ): 14.12  policy( 8.126 ): 20.7304

13  plan( 9 ): 7.84  policy( 8.172 ): 15.9986

14  plan( 8 ): 11.56  policy( 7.336 ): 18.7372

15  plan( 9 ): 14.0  policy( 7.09 ): 20.6456

17  plan( 9 ): 9.0  policy( 8.414 ): 16.9176
```

F MACRO LEVEL PLANNING TRACE

This appendix lists a part of the computation trace of the macro-level decision-theoretic planning process applied in the example of Section 5. We consider the states in the situation when the passenger is at check-in A121 with the next primary goal to go to passport control B. The trace shows the expected values of system's macro-level recommendation options for particular numbers of remaining time steps (10, 40, 50, 60, and 120 minutes) before the gate closes.

The first line of each paragraph shows: (1) the remaining time steps, (2) the assumed location, (3) already achieved secondary goals, and (4) the macro-level recommendation with the highest expected utility. After each headline follows a list of value-action pairs. For example, 123.45 look-for+Book+Dutyfree+Fastfood reads as the macro action *consider the secondary goals book, duty-free, and fast food* has an expected utility of 123.45.

```
10   Check-in A121, nil, none
      0.0   look-for+Batteries
      0.0   look-for+Batteries+Book
      0.0   look-for+Batteries+Book+Dutyfree
      0.0   look-for+Batteries+Book+Dutyfree+Fastfood
      0.0   look-for+Batteries+Book+Fastfood
      0.0   look-for+Batteries+Dutyfree
      0.0   look-for+Batteries+Dutyfree+Fastfood
      0.0   look-for+Batteries+Fastfood
      0.0   look-for+Book
      0.0   look-for+Book+Dutyfree
      0.0   look-for+Book+Dutyfree+Fastfood
      0.0   look-for+Book+Fastfood
      0.0   look-for+Dutyfree
      0.0   look-for+Dutyfree+Fastfood
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Batteries], none
      0.0   look-for+Book
      0.0   look-for+Book+Dutyfree
      0.0   look-for+Book+Dutyfree+Fastfood
      0.0   look-for+Book+Fastfood
      0.0   look-for+Dutyfree
      0.0   look-for+Dutyfree+Fastfood
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Book], none
      0.0   look-for+Batteries
      0.0   look-for+Batteries+Dutyfree
      0.0   look-for+Batteries+Dutyfree+Fastfood
      0.0   look-for+Batteries+Fastfood
      0.0   look-for+Dutyfree
      0.0   look-for+Dutyfree+Fastfood
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Dutyfree], none
      0.0   look-for+Batteries
      0.0   look-for+Batteries+Book
      0.0   look-for+Batteries+Book+Fastfood
      0.0   look-for+Batteries+Fastfood
      0.0   look-for+Book
      0.0   look-for+Book+Fastfood
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Fastfood], none
      0.0   look-for+Batteries
      0.0   look-for+Batteries+Book
      0.0   look-for+Batteries+Book+Dutyfree
      0.0   look-for+Batteries+Dutyfree
      0.0   look-for+Book
      0.0   look-for+Book+Dutyfree
      0.0   look-for+Dutyfree
      0.0   look-for-nothing
10   Check-in A121, [Batteries Book], none
      0.0   look-for+Dutyfree
      0.0   look-for+Dutyfree+Fastfood
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Book Dutyfree], none
      0.0   look-for+Batteries
      0.0   look-for+Batteries+Fastfood
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Dutyfree Fastfood], none
      0.0   look-for+Batteries
      0.0   look-for+Batteries+Book
      0.0   look-for+Book
      0.0   look-for-nothing
10   Check-in A121, [Batteries Book Dutyfree], none
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Batteries Dutyfree], none
      0.0   look-for+Book
      0.0   look-for+Book+Fastfood
      0.0   look-for+Fastfood
      0.0   look-for-nothing
10   Check-in A121, [Book Dutyfree Fastfood], none
      0.0   look-for+Batteries
      0.0   look-for-nothing
10   Check-in A121, [Book Fastfood], none
      0.0   look-for+Batteries
      0.0   look-for+Batteries+Dutyfree
      0.0   look-for+Dutyfree
      0.0   look-for-nothing
10   Check-in A121, [Batteries Book Dutyfree Fastfood], none
      0.0   look-for-nothing
10   Check-in A121, [Batteries Fastfood], none
      0.0   look-for+Book
      0.0   look-for+Book+Dutyfree
      0.0   look-for+Dutyfree
      0.0   look-for-nothing
10   Check-in A121, [Batteries Dutyfree Fastfood], none
      0.0   look-for+Book
      0.0   look-for-nothing
10   Check-in A121, [Batteries Book Fastfood], none
      0.0   look-for+Dutyfree
      0.0   look-for-nothing
```

```
...

40   Check-in A121, nil, look-for+Book
      ˜37.0, look-for+Batteries
      ˜40.0, look-for+Batteries+Book
      0.0, look-for+Batteries+Book+Dutyfree
      0.0, look-for+Batteries+Book+Dutyfree+Fastfood
      0.0, look-for+Batteries+Book+Fastfood
      0.0, look-for+Batteries+Dutyfree
      0.0, look-for+Batteries+Dutyfree+Fastfood
      0.0, look-for+Batteries+Fastfood
      84.1611, look-for+Book
      ˜39.0, look-for+Book+Dutyfree
      0.0, look-for+Book+Dutyfree+Fastfood
      ˜37.0, look-for+Book+Fastfood
      ˜36.0, look-for+Dutyfree
      0.0, look-for+Dutyfree+Fastfood
      ˜34.0, look-for+Fastfood
      66.5217, look-for-nothing
40   Check-in A121, [Batteries], look-for+Book
      148.161, look-for+Book
      ˜39.0, look-for+Book+Dutyfree
      0.0, look-for+Book+Dutyfree+Fastfood
      ˜37.0, look-for+Book+Fastfood
      ˜36.0, look-for+Dutyfree
      0.0, look-for+Dutyfree+Fastfood
      ˜34.0, look-for+Fastfood
      130.522, look-for-nothing
40   Check-in A121, [Book], look-for-nothing
      ˜37.0, look-for+Batteries
      0.0, look-for+Batteries+Dutyfree
      0.0, look-for+Batteries+Dutyfree+Fastfood
      0.0, look-for+Batteries+Fastfood
      ˜36.0, look-for+Dutyfree
      0.0, look-for+Dutyfree+Fastfood
      ˜34.0, look-for+Fastfood
      123.0, look-for-nothing
40   Check-in A121, [Dutyfree], look-for+Book
      ˜37.0, look-for+Batteries
      ˜40.0, look-for+Batteries+Book
      0.0, look-for+Batteries+Book+Fastfood
      0.0, look-for+Batteries+Fastfood
      134.161, look-for+Book
      ˜37.0, look-for+Book+Fastfood
      ˜34.0, look-for+Fastfood
      116.522, look-for-nothing
40   Check-in A121, [Fastfood], look-for+Book
      ˜37.0, look-for+Batteries
      ˜40.0, look-for+Batteries+Book
      0.0, look-for+Batteries+Book+Dutyfree
      0.0, look-for+Batteries+Dutyfree
      140.161, look-for+Book
      ˜39.0, look-for+Book+Dutyfree
      ˜36.0, look-for+Dutyfree
      122.522, look-for-nothing
40   Check-in A121, [Batteries Book], look-for-nothing
      ˜36.0, look-for+Dutyfree
      0.0, look-for+Dutyfree+Fastfood
      ˜34.0, look-for+Fastfood
      187.0, look-for-nothing
40   Check-in A121, [Book Dutyfree], look-for-nothing
      ˜37.0, look-for+Batteries
      0.0, look-for+Batteries+Fastfood
      ˜34.0, look-for+Fastfood
      173.0, look-for-nothing
40   Check-in A121, [Dutyfree Fastfood], look-for+Book
      ~37.0, look-for+Batteries
      ~40.0, look-for+Batteries+Book
      190.161, look-for+Book
      172.522, look-for-nothing
40   Check-in A121, [Batteries Book Dutyfree], look-for-nothing
      ~34.0, look-for+Fastfood
      237.0, look-for-nothing
40   Check-in A121, [Batteries Dutyfree], look-for+Book
      198.161, look-for+Book
      ~37.0, look-for+Book+Fastfood
      ~34.0, look-for+Fastfood
      180.522, look-for-nothing
40   Check-in A121, [Book Dutyfree Fastfood], look-for-nothing
      ~37.0, look-for+Batteries
      229.0, look-for-nothing
40   Check-in A121, [Book Fastfood], look-for-nothing
      ~37.0, look-for+Batteries
      0.0, look-for+Batteries+Dutyfree
      ~36.0, look-for+Dutyfree
      179.0, look-for-nothing
40   Check-in A121, [Batteries Book Dutyfree Fastfood], look-for-nothing
      293.0, look-for-nothing
40   Check-in A121, [Batteries Fastfood], look-for+Book
      204.161, look-for+Book
      ~39.0, look-for+Book+Dutyfree
      ~36.0, look-for+Dutyfree
      186.522, look-for-nothing
40   Check-in A121, [Batteries Dutyfree Fastfood], look-for+Book
      254.161, look-for+Book
      236.522, look-for-nothing
40   Check-in A121, [Batteries Book Fastfood], look-for-nothing
      ~36.0, look-for+Dutyfree
      243.0, look-for-nothing

...


60   Check-in A121, nil, look-for+Batteries+Book+Fastfood
      114.213, look-for+Batteries
      131.852, look-for+Batteries+Book
      99.2433, look-for+Batteries+Book+Dutyfree
      104.609, look-for+Batteries+Book+Dutyfree+Fastfood
      137.217, look-for+Batteries+Book+Fastfood
      96.517, look-for+Batteries+Dutyfree
      86.9692, look-for+Batteries+Dutyfree+Fastfood
      119.578, look-for+Batteries+Fastfood
      122.857, look-for+Book
      109.944, look-for+Book+Dutyfree
      115.309, look-for+Book+Dutyfree+Fastfood
      130.533, look-for+Book+Fastfood
      92.3043, look-for+Dutyfree
      97.6696, look-for+Dutyfree+Fastfood
      111.449, look-for+Fastfood
      105.217, look-for-nothing
60   Check-in A121, [Batteries], look-for+Book+Fastfood
      183.047, look-for+Book
      173.944, look-for+Book+Dutyfree
      179.309, look-for+Book+Dutyfree+Fastfood
      193.34, look-for+Book+Fastfood
      156.304, look-for+Dutyfree
      161.67, look-for+Dutyfree+Fastfood
      173.597, look-for+Fastfood
      163.304, look-for-nothing
```

```
60   Check-in A121, [Book], look-for+Batteries+Fastfood
      170.691, look-for+Batteries
      156.517, look-for+Batteries+Dutyfree
      143.447, look-for+Batteries+Dutyfree+Fastfood
      176.056, look-for+Batteries+Fastfood
      148.783, look-for+Dutyfree
      154.148, look-for+Dutyfree+Fastfood
      167.927, look-for+Fastfood
      161.696, look-for-nothing
60   Check-in A121, [Dutyfree], look-for+Batteries+Book+Fastfood
      140.139, look-for+Batteries
      158.347, look-for+Batteries+Book
      167.194, look-for+Batteries+Book+Fastfood
      149.377, look-for+Batteries+Fastfood
      149.352, look-for+Book
      158.199, look-for+Book+Fastfood
      139.901, look-for+Fastfood
      129.609, look-for-nothing
60   Check-in A121, [Fastfood], look-for+Batteries+Book
      164.574, look-for+Batteries
      182.782, look-for+Batteries+Book
      150.173, look-for+Batteries+Book+Dutyfree
      152.517, look-for+Batteries+Dutyfree
      173.786, look-for+Book
      160.873, look-for+Book+Dutyfree
      143.522, look-for+Dutyfree
      154.043, look-for-nothing
60   Check-in A121, [Batteries Book], look-for+Fastfood
      212.783, look-for+Dutyfree
      218.148, look-for+Dutyfree+Fastfood
      230.075, look-for+Fastfood
      219.783, look-for-nothing
60   Check-in A121, [Book Dutyfree], look-for+Batteries+Fastfood
      198.879, look-for+Batteries
      206.564, look-for+Batteries+Fastfood
      196.379, look-for+Fastfood
      186.087, look-for-nothing
60   Check-in A121, [Dutyfree Fastfood], look-for+Batteries+Book
      191.227, look-for+Batteries
      210.969, look-for+Batteries+Book
      198.178, look-for+Book
      178.435, look-for-nothing
60   Check-in A121, [Batteries Book Dutyfree], look-for+Fastfood
      254.466, look-for+Fastfood
      244.174, look-for-nothing
60   Check-in A121, [Batteries Dutyfree], look-for+Book+Fastfood
      207.439, look-for+Book
      217.731, look-for+Book+Fastfood
      197.988, look-for+Fastfood
      187.696, look-for-nothing
60   Check-in A121, [Book Dutyfree Fastfood], look-for+Batteries
      247.705, look-for+Batteries
      234.913, look-for-nothing
60   Check-in A121, [Book Fastfood], look-for+Batteries
      223.313, look-for+Batteries
      212.517, look-for+Batteries+Dutyfree
      203.522, look-for+Dutyfree
      210.522, look-for-nothing
60   Check-in A121, [Batteries Book Dutyfree Fastfood], look-for-nothing
      293.0, look-for-nothing
60   Check-in A121, [Batteries Fastfood], look-for+Book
      231.873, look-for+Book
      224.873, look-for+Book+Dutyfree
      205.13, look-for+Dutyfree
      212.13, look-for-nothing
```

```
60   Check-in A121, [Batteries Dutyfree Fastfood], look-for+Book
      256.265, look-for+Book
      236.522, look-for-nothing
60   Check-in A121, [Batteries Book Fastfood], look-for-nothing
      261.609, look-for+Dutyfree
      268.609, look-for-nothing

...


120   Check-in A121, nil, look-for+Batteries+Book+Fastfood
      118.009, look-for+Batteries
      137.752, look-for+Batteries+Book
      130.752, look-for+Batteries+Book+Dutyfree
      141.044, look-for+Batteries+Book+Dutyfree+Fastfood
      148.044, look-for+Batteries+Book+Fastfood
      111.009, look-for+Batteries+Dutyfree
      121.301, look-for+Batteries+Dutyfree+Fastfood
      128.301, look-for+Batteries+Fastfood
      124.96, look-for+Book
      117.96, look-for+Book+Dutyfree
      128.253, look-for+Book+Dutyfree+Fastfood
      135.253, look-for+Book+Fastfood
      98.2174, look-for+Dutyfree
      108.51, look-for+Dutyfree+Fastfood
      115.51, look-for+Fastfood
      105.217, look-for-nothing
120   Check-in A121, [Batteries], look-for+Book+Fastfood
      183.047, look-for+Book
      176.047, look-for+Book+Dutyfree
      186.34, look-for+Book+Dutyfree+Fastfood
      193.34, look-for+Book+Fastfood
      156.304, look-for+Dutyfree
      166.597, look-for+Dutyfree+Fastfood
      173.597, look-for+Fastfood
      163.304, look-for-nothing
120   Check-in A121, [Book], look-for+Batteries+Fastfood
      174.487, look-for+Batteries
      167.487, look-for+Batteries+Dutyfree
      177.78, look-for+Batteries+Dutyfree+Fastfood
      184.78, look-for+Batteries+Fastfood
      154.696, look-for+Dutyfree
      164.988, look-for+Dutyfree+Fastfood
      171.988, look-for+Fastfood
      161.696, look-for-nothing
120   Check-in A121, [Dutyfree], look-for+Batteries+Book+Fastfood
      142.4, look-for+Batteries
      162.143, look-for+Batteries+Book
      172.436, look-for+Batteries+Book+Fastfood
      152.693, look-for+Batteries+Fastfood
      149.352, look-for+Book
      159.644, look-for+Book+Fastfood
      139.901, look-for+Fastfood
      129.609, look-for-nothing
120   Check-in A121, [Fastfood], look-for+Batteries+Book
      166.835, look-for+Batteries
      186.578, look-for+Batteries+Book
      179.578, look-for+Batteries+Book+Dutyfree
      159.835, look-for+Batteries+Dutyfree
      173.786, look-for+Book
      166.786, look-for+Book+Dutyfree
      147.043, look-for+Dutyfree
      154.043, look-for-nothing
120   Check-in A121, [Batteries Book], look-for+Fastfood
      212.783, look-for+Dutyfree
      223.075, look-for+Dutyfree+Fastfood
      230.075, look-for+Fastfood
      219.783, look-for-nothing
120   Check-in A121, [Book Dutyfree], look-for+Batteries+Fastfood
      198.879, look-for+Batteries
      209.171, look-for+Batteries+Fastfood
      196.379, look-for+Fastfood
      186.087, look-for-nothing
120   Check-in A121, [Dutyfree Fastfood], look-for+Batteries+Book
      191.227, look-for+Batteries
      210.969, look-for+Batteries+Book
      198.178, look-for+Book
      178.435, look-for-nothing
120   Check-in A121, [Batteries Book Dutyfree], look-for+Fastfood
      254.466, look-for+Fastfood
      244.174, look-for-nothing
120   Check-in A121, [Batteries Dutyfree], look-for+Book+Fastfood
      207.439, look-for+Book
      217.731, look-for+Book+Fastfood
      197.988, look-for+Fastfood
      187.696, look-for-nothing
120   Check-in A121, [Book Dutyfree Fastfood], look-for+Batteries
      247.705, look-for+Batteries
      234.913, look-for-nothing
120   Check-in A121, [Book Fastfood], look-for+Batteries
      223.313, look-for+Batteries
      216.313, look-for+Batteries+Dutyfree
      203.522, look-for+Dutyfree
      210.522, look-for-nothing
120   Check-in A121, [Batteries Book Dutyfree Fastfood], look-for-nothing
      293.0, look-for-nothing
120   Check-in A121, [Batteries Fastfood], look-for+Book
      231.873, look-for+Book
      224.873, look-for+Book+Dutyfree
      205.13, look-for+Dutyfree
      212.13, look-for-nothing
120   Check-in A121, [Batteries Dutyfree Fastfood], look-for+Book
      256.265, look-for+Book
      236.522, look-for-nothing
120   Check-in A121, [Batteries Book Fastfood], look-for-nothing
      261.609, look-for+Dutyfree
      268.609, look-for-nothing
```

G SHOPPING GUIDE POLICIES

This appendix contains extracts of the policies that were applied in the user studies of Section 7.2 and 7.3. The syntax is in the style of the tcl/tk programming language, such that the policy can directly be applied by a tcl/tk interpreter, which we employed to visualize the navigation recommendations on the handheld device. A policy file for the first user study is of a size less than 100 kbyte and less than about 1500 lines long.

```
set statetable(0100:n:0:0:0:0:0) "ShowPic stay_on_upper_floor.gif"
set statetable(0200:n:0:0:0:0:0) "ShowPic turn_right.gif"
set statetable(0300:n:0:0:0:0:0) "ShowPic look_for_items.gif;
                                  EnableButtonList { 1 1 0 0 0}"
set statetable(0300:b:1:0:0:0:0) "ShowPic turn_left.gif"
set statetable(0300:b:0:1:0:0:0) "ShowPic turn_left.gif"
set statetable(0300:b:1:1:0:0:0) "ShowPic turn_left.gif"
set statetable(0300:f:0:0:0:0:0) "ShowPic turn_left.gif"
set statetable(0300:h:1:0:0:0:0) "ShowPic stand_in_line.gif"
set statetable(0300:h:0:1:0:0:0) "ShowPic stand_in_line.gif"
set statetable(0300:h:1:1:0:0:0) "ShowPic stand_in_line.gif"
set statetable(0400:n:1:1:0:0:0) "ShowPic look_for_items.gif;
                                  EnableButtonList { 0 0 0 0 1}"
set statetable(0400:n:1:0:0:0:0) "ShowPic look_for_items.gif;
                                  EnableButtonList { 0 0 0 0 1}"
set statetable(0400:n:0:0:0:0:0) "ShowPic look_for_items.gif;
                                  EnableButtonList { 0 0 0 0 1}"
set statetable(0400:n:0:1:0:0:0) "ShowPic look_for_items.gif;
                                  EnableButtonList { 0 0 0 0 1}"
set statetable(0400:b:1:1:0:0:1) "ShowPic turn_left.gif"
set statetable(0400:f:1:1:0:0:0) "ShowPic turn_left.gif"
set statetable(0400:h:1:1:0:0:1) "ShowPic stand_in_line.gif"
set statetable(0400:b:1:0:0:0:1) "ShowPic turn_left.gif"
set statetable(0400:f:1:0:0:0:0) "ShowPic turn_left.gif"
set statetable(0400:h:1:0:0:0:1) "ShowPic stand_in_line.gif"
set statetable(0400:b:0:0:0:0:1) "ShowPic turn_left.gif"
set statetable(0400:f:0:0:0:0:0) "ShowPic turn_left.gif"
set statetable(0400:h:0:0:0:0:1) "ShowPic stand_in_line.gif"
set statetable(0400:b:0:1:0:0:1) "ShowPic turn_left.gif"
set statetable(0400:f:0:1:0:0:0) "ShowPic turn_left.gif"
set statetable(0400:h:0:1:0:0:1) "ShowPic stand_in_line.gif"

...
```

```
set statetable(3000:n:1:1:1:1:1) "ShowPic stop.gif"
```

Each state has the following features: (1) The ID of a location; e.g., 0400 corresponds to Location 4, (2) the information about the status of the shopping action, which can be n (normal, i.e., the user is either outside any shops or he is inside a shop but has not yet found any item), h (hit, i.e., the user has found one or several items in a shop but has not paid the item(s), yet), f (failure, i.e., the user has indicated that he has not found any item that he is looking for in the current shop), and b (bought, i.e., the user has paid the item(s) that he has found in the current shop). The status feature is followed by five features each corresponding to the information if a particular item from the shopping list has already been bought or not (1 means item X, e.g., bread, has been bought, 0 means item X has not been bought, yet). This completes the state information. According to the location, the status, and the information about which of the items on the shopping list have already been bought, the system can give the next recommendation. In the user study, this corresponded to either showing only a picture (e.g., turn_left.gif) or asking the user to look for shopping items that the system expects to be available in the current shop and enabling the corresponding check boxes in the shopping list (which is shown on the handheld display).

Time-Dependent Finite-Horizon Policy To deal with a fixed deadline, we have employed a special form of finite-horizon decision-theoretic planning for the second user study. The resulting policy is time-dependent. Each time-dependent recommendation is best read backwards: For example, the first line reads as follows: *With more than 430 seconds remaining until the deadline, the recommendation is to turn left; with less than 431 seconds remaining until the deadline, the recommendation is to go straight.* A policy file for the second user study is of a size less than 300 kbyte and less than 3000 lines long.

```
set statetable(0100:n:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"go_straight.gif"
    431:"turn_left.gif"  )
set statetable(0200:n:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_left.gif"
    361:"look_for_items.gif"  ; EnableButtonList { 0 0 0 1 0 0 } )
set statetable(0200:b:0:0:0:1:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_left.gif"  )
set statetable(0200:f:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_left.gif"  )
set statetable(0200:h:0:0:0:1:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"stand_in_line.gif"  )
set statetable(0300:n:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"halbturn_left.gif"  )
set statetable(0300:n:0:0:0:1:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"halbturn_left.gif"  )
set statetable(0400:n:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_left.gif"  )
set statetable(0400:n:0:0:0:1:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_left.gif"  )
set statetable(0500:n:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"go_straight.gif"
    409:"turn_right.gif"  738:"go_straight.gif"  899:"turn_right.gif"  )
set statetable(0500:n:0:0:0:1:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"go_straight.gif"
    357:"turn_right.gif"  686:"go_straight.gif"  847:"turn_right.gif"  )
```

```
set statetable(0600:n:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_right.gif"
    404:"look_for_items.gif"  ; EnableButtonList { 0 0 0 0 1 0 }
    736:"turn_right.gif"
    894:"look_for_items.gif"  ; EnableButtonList { 0 0 0 0 1 0 } )
set statetable(0600:n:0:0:0:1:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_right.gif"
    352:"look_for_items.gif"  ; EnableButtonList { 0 0 0 0 1 0 }
    684:"turn_right.gif"
    842:"look_for_items.gif"  ; EnableButtonList { 0 0 0 0 1 0 } )
set statetable(0600:b:0:0:0:0:1:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_right.gif"  )
set statetable(0600:f:0:0:0:0:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_right.gif"  )
set statetable(0600:h:0:0:0:0:1:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"stand_in_line.gif"  )
set statetable(0600:b:0:0:0:1:1:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_right.gif"  )
set statetable(0600:f:0:0:0:1:0:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"turn_right.gif"  )
set statetable(0600:h:0:0:0:1:1:0)
    recommendation_at_time_t( 0:"stop.gif"  1:"stand_in_line.gif"  )

...
```

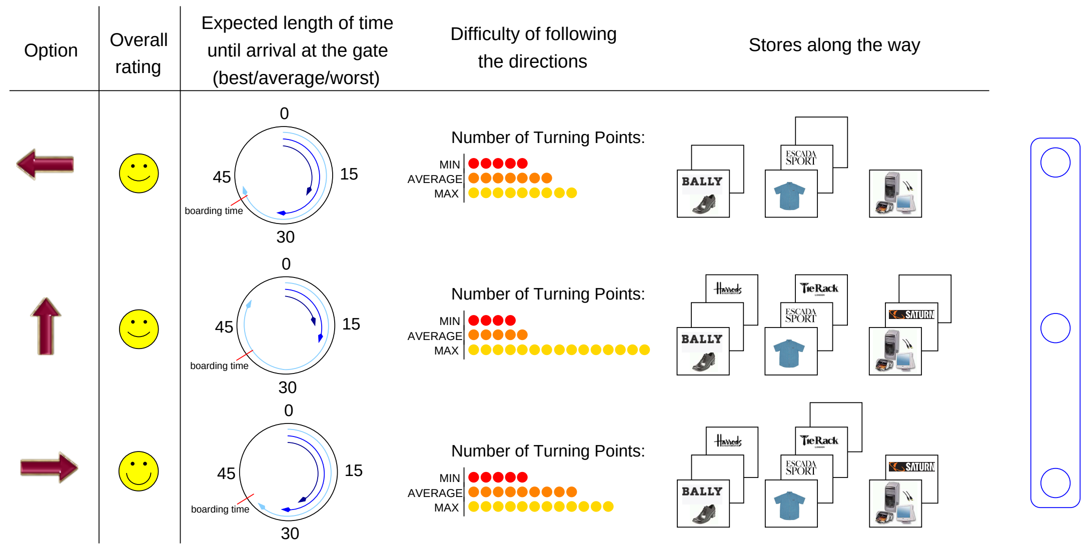
H QUESTIONNAIRE TO ELICIT USEFUL INFOMATION FOR EXPLANATIONS

This appendix summarizes the complete questionnaire of the user study described in Section 7.4. The questionnaire was used to elict information that users of a decision-theoretic shopping guide might find useful to understand the recommendations given by the system.

The questionnaire includes six sheets alltogether. On the first three sheeds, the subjects were given assessments of three different route recommendations. They were asked to select one and explain/comment on their selection. On each of the first three sheets, there was half a page of space for a justification of the subject's selection. Each of the tables in Figures H.1 and H.2 shows three route recommendations with the explanations that were used in the study.

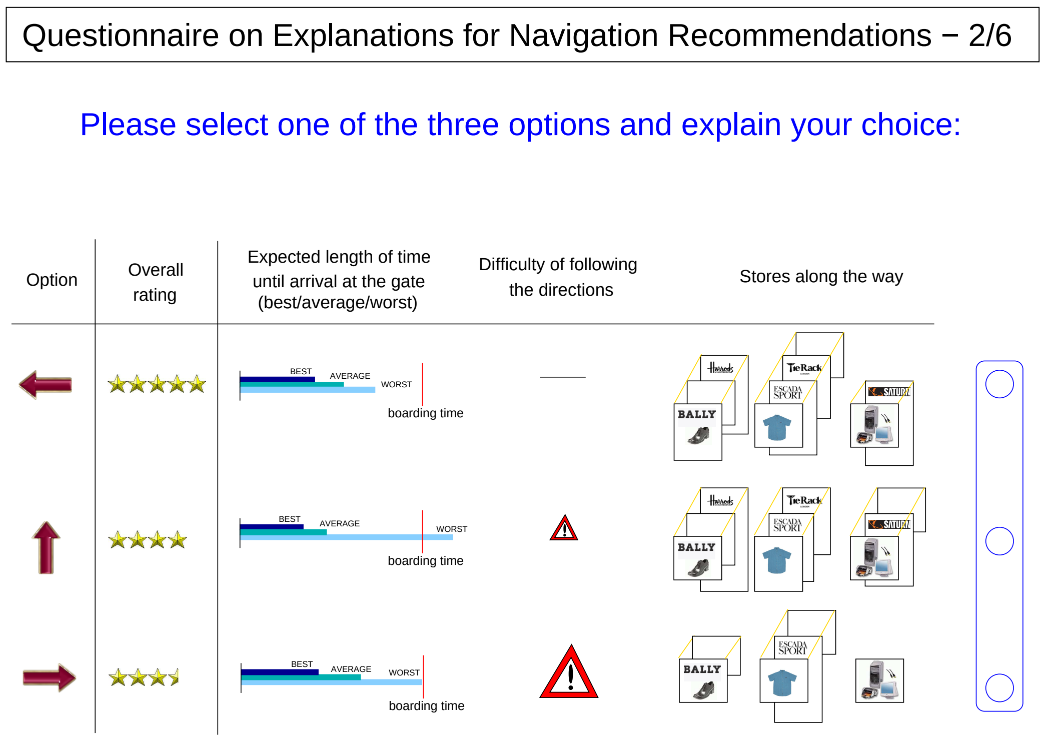
Questionnaire on Explanations for Navigation Recommendations − 1/6

Please select one of the three options and explain your choice:



Option
Overall rating
Expected length of time until arrival at the gate (best/average/worst)
Difficulty of following the directions
Stores along the way
0
15
30
45
boarding time
Number of Turning Points:
MIN
AVERAGE
MAX
BALLY
ESCADA SPORT
Harrods
TieRack
SATURN

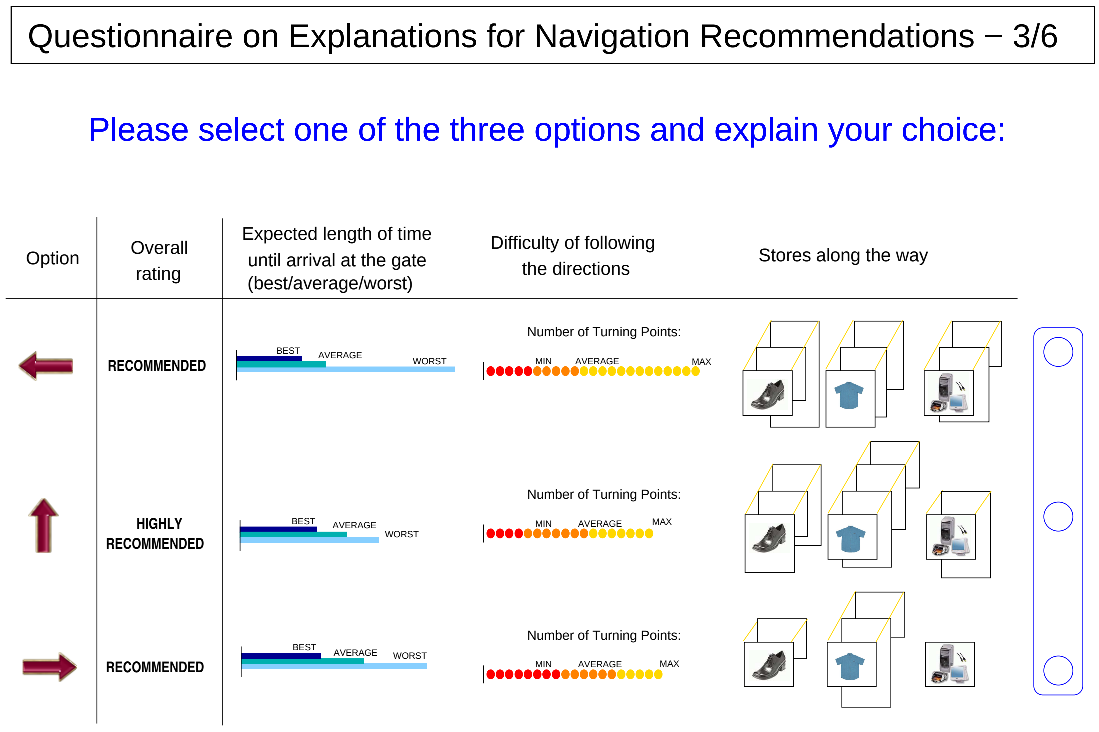
***Figure H.1**: Questionnaire: Select one of three options—first example.*
(For each possible direction, the table used in the questionnaire gives an overall rating, the expected length of time until arrival at the gate, the difficulty of following the direction, and information about stores along the way. Based on this information, the subjects were supposed to decide for one direction and explain/comment on their choice.)



Questionnaire on Explanations for Navigation Recommendations − 2/6

Please select one of the three options and explain your choice:

Option	Overall rating	Expected length of time until arrival at the gate (best/average/worst)	Difficulty of following the directions	Stores along the way	
←	★★★★★	BEST AVERAGE WORST boarding time	—	Harrods TieRack ESCADA SPORT BALLY SATURN	○
↑	★★★★	BEST AVERAGE WORST boarding time	⚠	Harrods TieRack ESCADA SPORT BALLY SATURN	○
→	★★★½	BEST AVERAGE WORST boarding time	⚠	BALLY ESCADA SPORT	○



Questionnaire on Explanations for Navigation Recommendations − 3/6

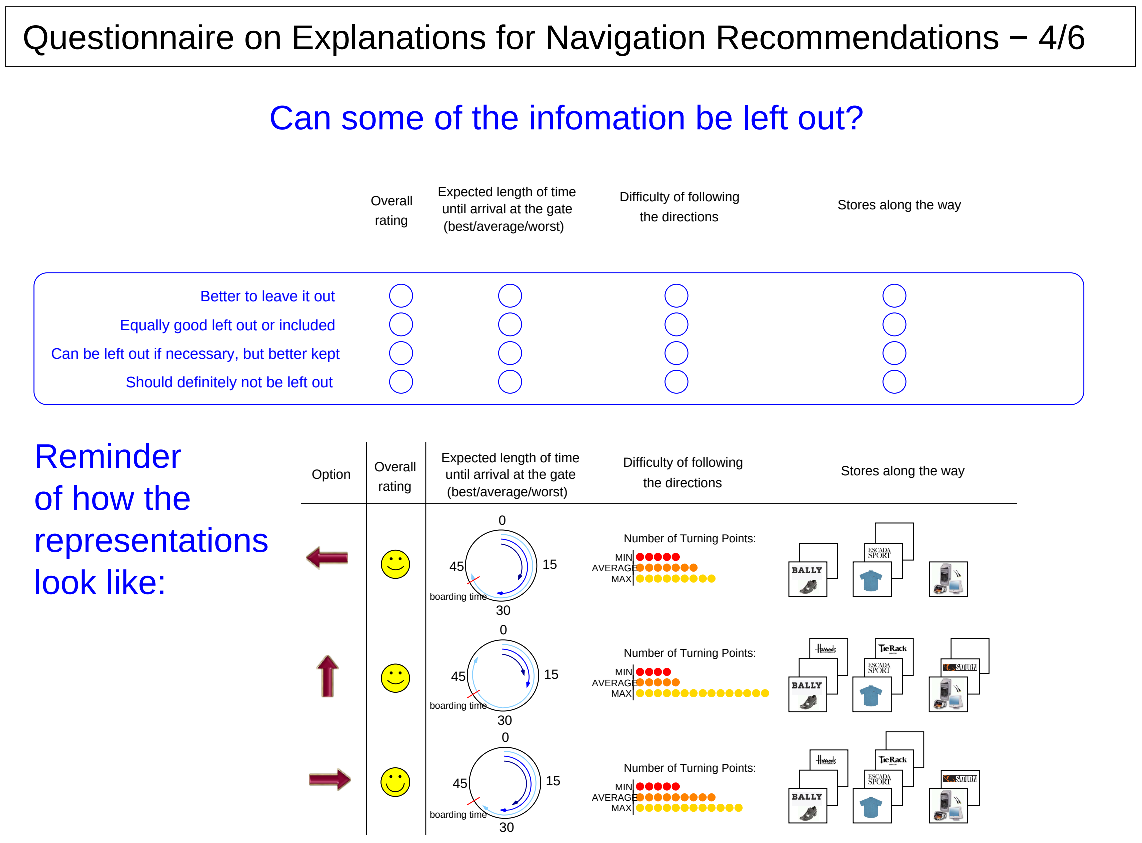
Please select one of the three options and explain your choice:

Option	Overall rating	Expected length of time until arrival at the gate (best/average/worst)	Difficulty of following the directions	Stores along the way	
←	RECOMMENDED	BEST AVERAGE WORST	Number of Turning Points: MIN AVERAGE MAX		○
↑	HIGHLY RECOMMENDED	BEST AVERAGE WORST	Number of Turning Points: MIN AVERAGE MAX		○
→	RECOMMENDED	BEST AVERAGE WORST	Number of Turning Points: MIN AVERAGE MAX		○

***Figure H.2**: Questionnaire: Select one of three options—second and third example.*

On the fourth sheet of the questionnaire, the subjects were reminded of the differents explanations that were used in the first three sheets to explain what the user can expect when following any of the directions. The subjects were asked to examine the information that was given in each column of the table and to assess how useful each of the pieces of information is, by tagging any of four options ranging from “This piece of information should better be left out” to “This piece of information should definitely not be left out”. For their assessment, the subjects should also take into account the appropriateness of the representation of the information given. Figure H.3 shows one of the three tables that the subjects were asked to assess.

The last two sheets addressed the representation of information. Figure H.4 shows the three representations of the overall rating of a navigation recommendation; Figure H.5 shows three representations of the expected length of time until arrival at the gate; Figure H.6 shows three representations of the difficulty of following upcoming directions; and Figure H.7 shows three representations of information on stores along the way. Each form of representation could be rated on a scale with seven grades ranging from “not useful at all” to “very useful”.



Questionnaire on Explanations for Navigation Recommendations − 4/6

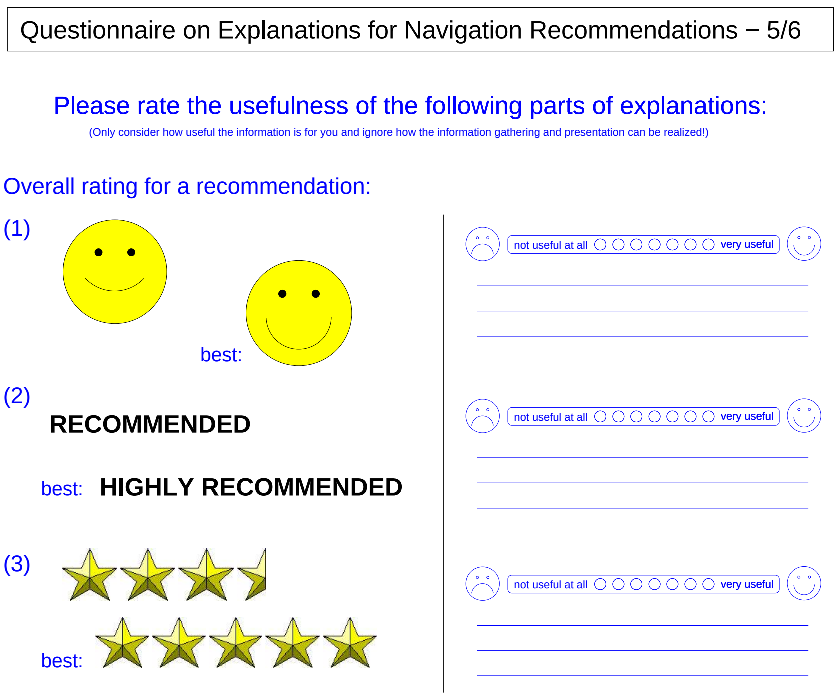
Can some of the infomation be left out?

	Overall rating	Expected length of time until arrival at the gate (best/average/worst)	Difficulty of following the directions	Stores along the way
Better to leave it out	○	○	○	○
Equally good left out or included	○	○	○	○
Can be left out if necessary, but better kept	○	○	○	○
Should definitely not be left out	○	○	○	○

Reminder of how the representations look like:

Option	Overall rating	Expected length of time until arrival at the gate (best/average/worst)	Difficulty of following the directions	Stores along the way
⬅	☺	0 15 30 45 boarding time	Number of Turning Points: MIN ●●●●● AVERAGE ●●●●●●● MAX ●●●●●●●●●	BALLY ESCADA SPORT
⬆	☺	0 15 30 45 boarding time	Number of Turning Points: MIN ●●●● AVERAGE ●●●●● MAX ●●●●●●●●●●●●●●●	Harrods BALLY TieRack ESCADA SPORT SATURN
➡	☺	0 15 30 45 boarding time	Number of Turning Points: MIN ●●●●● AVERAGE ●●●●●●●●● MAX ●●●●●●●●●●●●	Harrods BALLY TieRack ESCADA SPORT SATURN

***Figure H.3**: Questionnaire: Can some of the explanatory information be left out?*
(After being reminding of the tables used on the fi rst three sheets, the subjects were asked how useful they have found the information characterizing the different options. For each type of information, they could quote if they prefered some information to be left out or not. The students were asked to take also the form of representation of the information into account.)



Questionnaire on Explanations for Navigation Recommendations − 5/6

Please rate the usefulness of the following parts of explanations:

(Only consider how useful the information is for you and ignore how the information gathering and presentation can be realized!)

Overall rating for a recommendation:

(1)

best:

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

(2)

RECOMMENDED

best: HIGHLY RECOMMENDED

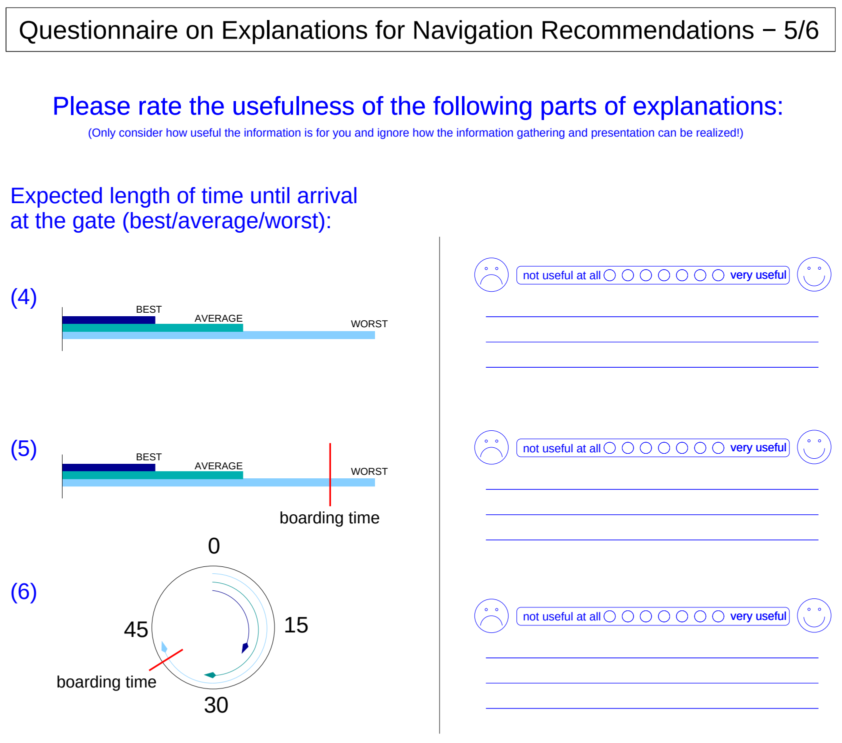
not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

(3)

best:

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

Figure H.4: *Three representations of the overall rating of a navigation recommendation.*



Questionnaire on Explanations for Navigation Recommendations − 5/6

Please rate the usefulness of the following parts of explanations:

(Only consider how useful the information is for you and ignore how the information gathering and presentation can be realized!)

Expected length of time until arrival at the gate (best/average/worst):

(4)

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

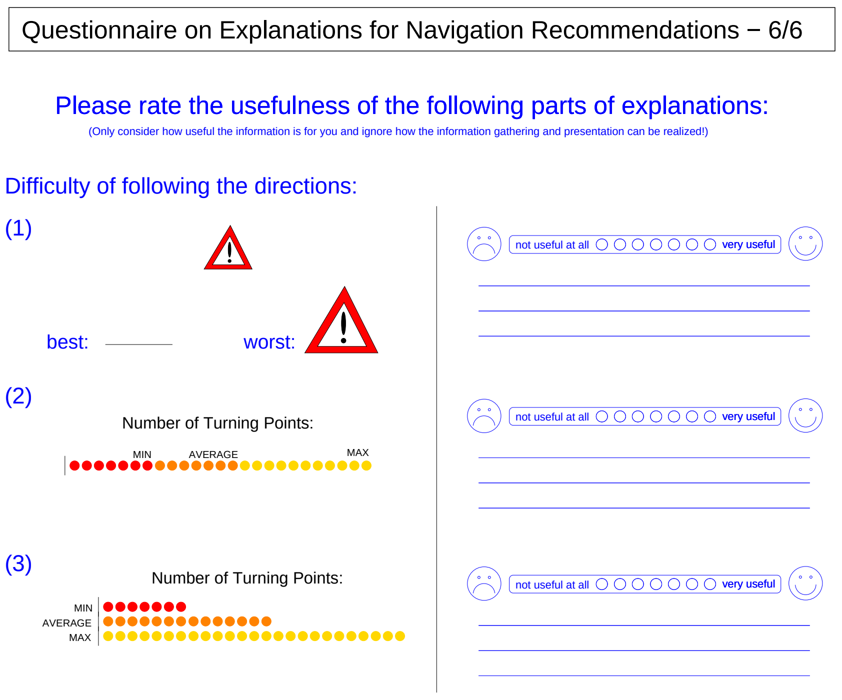
(5)

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

(6)

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

Figure H.5: *Three representations of the expected length of time until arrival at the gate.*



Questionnaire on Explanations for Navigation Recommendations − 6/6

Please rate the usefulness of the following parts of explanations:

(Only consider how useful the information is for you and ignore how the information gathering and presentation can be realized!)

Difficulty of following the directions:

(1) best: worst:

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

(2) Number of Turning Points:

MIN AVERAGE MAX

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

(3) Number of Turning Points:

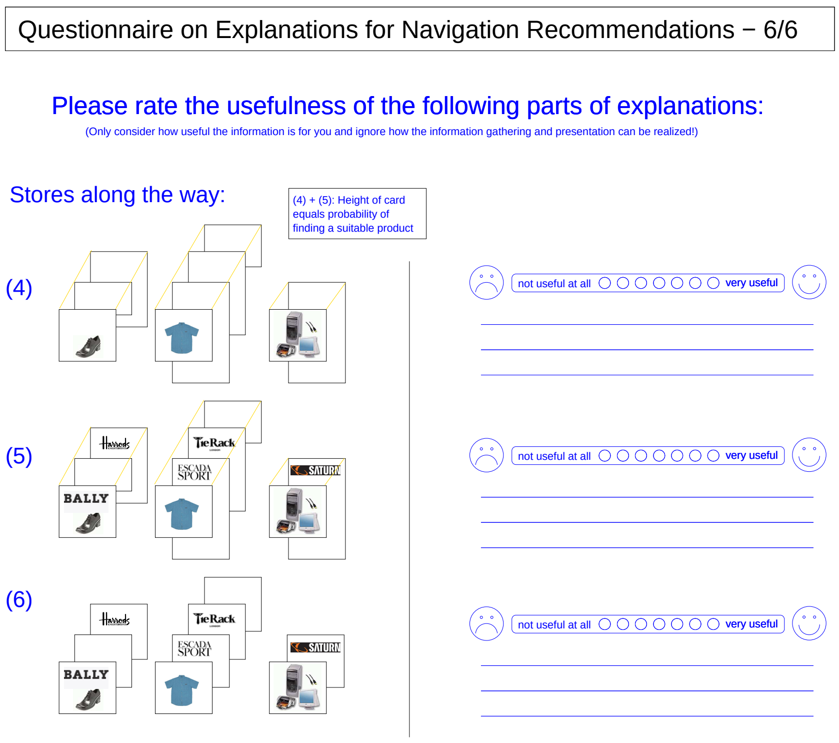
MIN

AVERAGE

MAX

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

***Figure H.6**: Three representations of the diffi culty of following upcoming direction.*



Questionnaire on Explanations for Navigation Recommendations − 6/6

Please rate the usefulness of the following parts of explanations:

(Only consider how useful the information is for you and ignore how the information gathering and presentation can be realized!)

Stores along the way:

(4) + (5): Height of card equals probability of finding a suitable product

(4)

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

(5)

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

(6)

not useful at all ○ ○ ○ ○ ○ ○ ○ very useful

***Figure H.7**: Three representations of information on stores along the way.*

I VIRTUAL BEACONS IN THE USER STUDY OF SECTION 7.3

Figures I.1 and I.2 comprise the complete register of all virtual beacons that the experimenter simulated in the user study described in Section 7.3. Each trial started at virtual beacon number 1 (one of the mall entrances) and ended at virtual beacon number 36 (one of the mall exits).

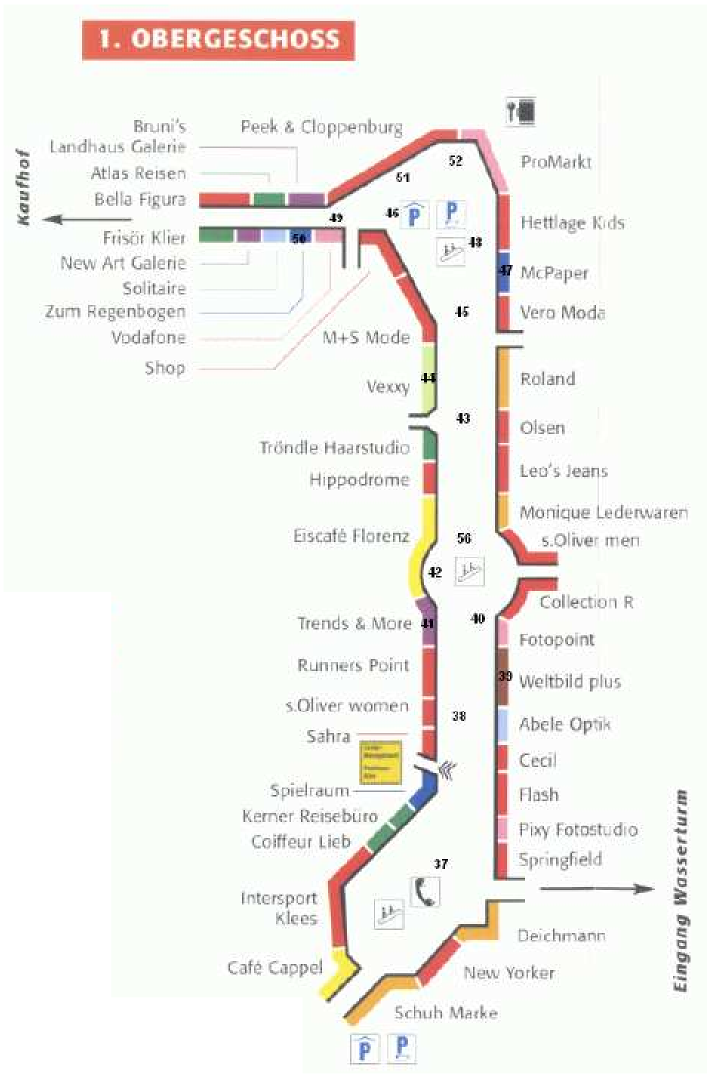


Figure I.1: *Virtual beacons on the first floor of the shopping mall.*

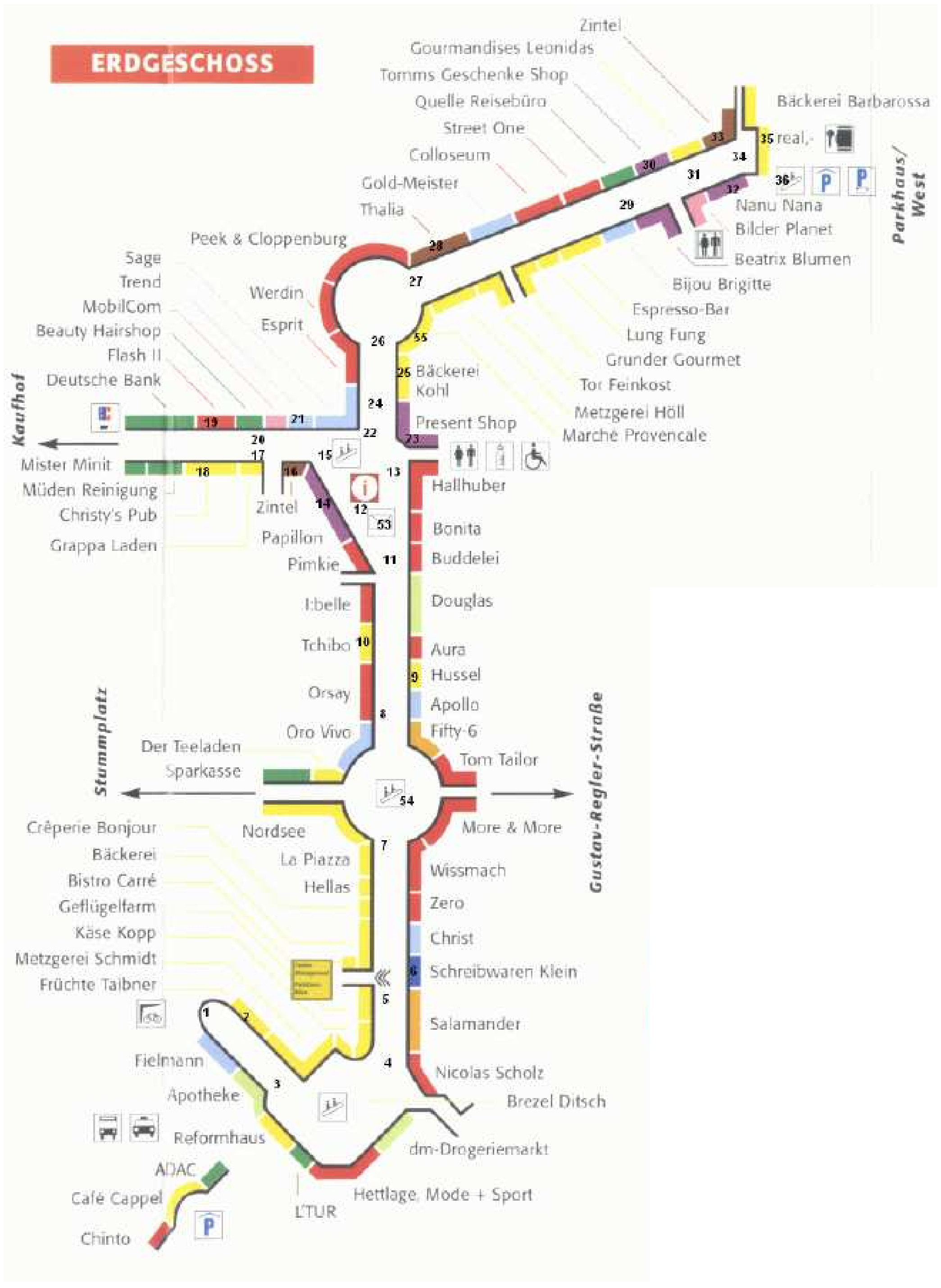


Figure I.2: *Virtual beacons in the basement of the shopping mall.*

Bibliography

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, *26*(11), 832–843.

Alonso, F., Maté, L., Juristo, N., Muñoz, P. L., & Pazos, J. (1994). Applying metrics to machine-learning tools: a knowledge engineering approach. *AI Magazine*, *15*(3), 63–75.

André, E. (1995). *Ein planbasierter Ansatz zur Generierung multimedialer Präsentationen.* Berlin: AKA Verlag.

André, E., M¨uller, J., & Rist, T. (1996). WIP/PPP: Automatic generation of personalized multimedia presentations. In *MULTIMEDIA'96: Proceedings of the Fourth ACM Multimedia Conference* (pp. 407–408). New York, NY, USA: ACM Press.

Anegg, H., Kunczier, H., Michlmayr, E., Pospischil, G., & Umlauft, M. (2002). LoL@: Designing a location based UMTS application. *Elektrotechnik und Informationstechnik*, *119*(2), 48–51.

Armstrong, R., Freitag, D., Joachims, T., & Mitchell, T. (1995). WebWatcher: A learning apprentice for the World Wide Web. In *AAAI Spring Symposium on Information Gathering From Heterogeneous, Distributed Environments.*

Asthana, A., Cracatts, M., & Krzyzanowski. (1994). An indoor wireless system for personalized shopping assistance. In *Proceedings of the First IEEE Workshop on Mobile Computing Systems and Applications.* Santa Cruz, CA.

Astr¨om, K. J. (1965). Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, *10*, 174–205.

Baader, F., & Hollunder, B. (1991). KRIS: Knowledge representation and inference system. *ACM SIGART Bulletin*, *2*(3), 8–14.

Babaian, T., Grosz, B. J., & Shieber, S. M. (2002). A writer's collaborative assistant. In Y. Gil & D. B. Leake (Eds.), *IUI 2002: Proceedings of the 7th International Conference on Intelligent User Interfaces* (pp. 7–14). New York: ACM Press.

Bauer, M. (1996). Acquisition of user preferences for plan recognition. In S. Carberry & I. Zukerman (Eds.), *Proceedings of the Fifth International Conference on User Modeling* (pp. 105–112). Boston, MA: User Modeling, Inc.

Bauer, M., Biundo, S., Dengler, D., Koehler, J., & Paul, G. (1993). PHI – A logic-based tool for intelligent help systems. In *IJCAI'93: Proceedings of the 13th International Joint Conference on Artificial Intelligence* (pp. 460–466). Chambery, France: Morgan Kaufmann.

Baus, J. (2003). *Ressourcenadaptierende hybride Navigation.* Berlin: AKA Verlag.

Baus, J., Kr¨uger, A., & Wahlster, W. (2002). A resource-adaptive mobile navigation system. In Y. Gil & D. B. Leake (Eds.), *IUI 2002: Proceedings of the 7th International Conference on Intelligent User Interfaces* (pp. 15–22). New York: ACM Press.

Bellman, R. (1957). *Dynamic programming.* Princeton, New Jersey: Princeton University Press.

Bertsekas, D. P. (1987). *Dynamic programming: Deterministic and stochastic models.* Englewood Cliffs, NJ: Prentice-Hall.

Bertsekas, D. P., & Tsitsiklis, J. N. (1989). *Parallel and distributed computation: Numerical methods.* Englewood Cliffs NJ: Prentice Hall.

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1–2), 281–300.

Bohnenberger, T. (2000). How can an intelligent dialog system plan ahead? In M. M¨uller (Ed.), *ABIS-00, Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen [ABIS-00, Adaptivity and User Modeling in Interactive Software Systems].* Osnabr¨uck, Germany: University of Osnabr¨uck.

Bohnenberger, T. (2001). Dialog strategies for adaptive help systems,. In N. Henze (Ed.), *ABIS-01, Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen [ABIS-01, Adaptivity and User Modeling in Interactive Software Systems].* Dortmund, Germany: University of Dortmund.

Bohnenberger, T. (2004). *Decision-theoretic planning for user-adaptive systems: Dealing with multiple goals and resource limitations.* Unpublished doctoral dissertation, Computer Science Department, Saarland University, Saarbr¨ucken, Germany. (This thesis)

Bohnenberger, T., Brandherm, B., Großmann-Hutter, B., Heckmann, D., & Wittig, F. (2002). Empirically grounded decision-theoretic adaptation to situation-dependent resource limitations. *Künstliche Intelligenz*, *16*(3), 10–16.

Bohnenberger, T., & Butz, A. (2000). Decision-theoretic planning of navigation instructions. In R. Malaka (Ed.), *ECAI 2000 Workshop on Artificial Intelligence in Mobile Systems* (pp. 7–12). Berlin.

Bohnenberger, T., Jacobs, O., & Jameson, A. (2005). *DTP meets user requirements: Enhancements and studies of an intelligent shopping guide.* (Manuscript submitted for publication)

Bohnenberger, T., & Jameson, A. (2001). When policies are better than plans: Decision-theoretic planning of recommendation sequences. In J. Lester (Ed.), *IUI 2001: International Conference on Intelligent User Interfaces* (pp. 21–24). New York: ACM.

Bohnenberger, T., Jameson, A., Kr¨uger, A., & Butz, A. (2002a). Location-aware shopping assistance: Evaluation of a decision-theoretic approach. In F. Paterno (Ed.), *Proceedings of the Fourth International Symposium on Human-Computer Interaction with Mobile Devices* (pp. 155–169). Pisa.

Bohnenberger, T., Jameson, A., Kr¨uger, A., & Butz, A. (2002b). User acceptance of a decision-theoretic, location-aware shopping guide. In Y. Gil & D. B. Leake (Eds.), *IUI 2002: International Conference on Intelligent User Interfaces* (pp. 178–179). New York: ACM.

Bohnenberger, T., & Kr¨uger, A. (2001). Combining predefined routes and decision-theoretic planning to generate adaptive navigation recommendations. In R. Malaka & A. Kr¨uger (Eds.), *Proceedings of the AIMS-2001 Workshop: Artificial Intelligence in Mobile Systems.* Seattle, Washington.

Boutilier, C., Brafman, R. I., & Geib, C. (1997). Prioritized goal decomposition of Markov decision processes: Toward a synthesis of classical and decision-theoretic planning. In *IJCAI'97: Proceedings of the 15th International Joint Conference on Artificial Intelligence* (pp. 1156–1162). San Francisco: Morgan Kaufmann Publishers.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1–94.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. In C. S. Mellish (Ed.), *IJCAI'95: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1104–1111). San Mateo, CA: Morgan Kaufmann.

Boutilier, C., & Poole, D. (1996). Computing optimal policies for partially observable decision processes using compact representations. In W. J. Clancey & D. Weld (Eds.), *AAAI'96: Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 1168–1175). Portland, OR.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In B. Nebel (Ed.), *IJCAI 2001: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (pp. 690–700). Seattle, WA: Morgan Kaufmann.

Boutilier, C., Reiter, R., Soutchanski, M., & Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In H. Kautz & B. Porter (Eds.), *AAAI 2000: Proceedings of the Seventeenth National Conference on Artificial Intelligence* (pp. 355–362). Austin, Texas: AAAI Press.

Boyan, J., & Littman, M. (2000). Exact solutions to time-dependent MDPs. In *Proceedings of the Workshop "Beyond MDPs: Representations and Algorithms" at the Sixteenth Conference on Uncertainty in Artificial Intelligence.*

Boyen, X., & Koller, D. (1998). Tractable inference for complex stochastic processes. In G. F. Cooper & S. Moral (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference* (pp. 33–42). San Francisco: Morgan Kaufmann.

Brafman, R. (1997). A heuristic variable grid solution method for POMDPs. In B. Kuipers & B. Webber (Eds.), *AAAI'97: Proceedings of the Fourteenth National Conference on Artificial Intelligence* (pp. 727–733). Providence, Rhode Island.

Brafman, R., Heckerman, D., & Shani, G. S. (2003). Recommendation as a stochastic sequential decision problem. In E. Giunchiglia, N. Muscettola, & D. S. Nau (Eds.), *ICAPS 2003: Proceedings of the International Conference on Automated Planning and Scheduling* (pp. 164–173). Trento, Italy.

Brafman, R. I., Heckerman, D., & Shani, G. (2004). An MDP-based recommender system. *Journal of Machine Learning Research*. (In press.)

Brandherm, B. (2003). An extension of the differential approach for Bayesian network inference to dynamic Bayesian networks. In I. Russell & S. Haller (Eds.), *Proceedings of the Sixteenth International FLAIRS conference* (pp. 486–490). Menlo Park, California: AAAI Press.

Brusilovsky, P., & Cooper, D. W. (2002). Domain, task, and user models for an adaptive hypermedia performance support system. In Y. Gil & D. B. Leake (Eds.), *IUI 2002: Proceedings of the 7th International Conference on Intelligent User Interfaces* (pp. 23–30). New York: ACM Press.

Bui, H. H., Venkatesh, S., & West, G. (2000). On the recognition of abstract markov policies. In *AAAI 2000: Proceeding of the Seventeenth National Conference on Artificial Intelligence* (pp. 524–530). Menlo Park, CA: AAAI Press.

Butz, A., Baus, J., Kr¨uger, A., & Lohse, M. (2001). A hybrid indoor navigation system. In J. Lester (Ed.), *IUI 2001: International Conference on Intelligent User Interfaces* (pp. 25–32). New York: ACM.

Cassandra, A., Littman, M., & Zhang, N. (1997). Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In D. Geiger & P. P. Shenoy (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Thirteenth Conference* (pp. 54–61). San Francisco: Morgan Kaufmann.

Cawsey, A. (1993). Planning interactive explanations. *International Journal of Man-Machine Studies*, *38*, 169–199.

Chin, D. N. (1989). KNOME: Modeling what the user knows in UC. In A. Kobsa & W. Wahlster (Eds.), *User Models in Dialog Systems* (pp. 74–107). Berlin: Springer.

Clark, H. H., & Brennan, S. E. (1991). Grounding in communication. In L. B. Resnick, J. M. Levine, & S. D. Teasley (Eds.), *Perspectives on socially shared cognition* (pp. 127–149). Washington, D.C.: American Psychological Association.

Cypher, A. (1991). Eager: Programming repetitive tasks by example. In *Proceedings of CHI'91 Conference on Human Factors in Computing Systems* (pp. 33–39). New York: ACM Press.

Darken, R. P., & Peterson, B. (2001). Spatial orientation, wayfinding, and representation. In K. Stanney (Ed.), *Handbook of virtual environment technology.* Mahwah, NJ: Erlbaum.

Davies, J. R., Gertner, A. S., Lesh, N., Rich, C., Sidner, C. L., & Rickel, J. (2001). Incorporating tutorial strategies into an intelligent assistant. In J. Lester (Ed.), *IUI 2001: Proceedings of the 6th International Conference on Intelligent User Interfaces* (pp. 53–56). New York: ACM Press.

Dean, T., Kaelbling, L. P., Kirman, J., & Nicholson, A. (1995). Planning under time constraints in stochastic domains. *Artificial Intelligence*, *76*(1–2), 35–74.

Dean, T., & Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. In *IJCAI'95: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1121–1129). Montreal, Quebec, Canada.

Dempster, A. P. (1968). A generalization of Bayesian inference. *Journal of the Royal Statistical Society, Series B*, *30*, 205–247. (With discussion)

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Drake, A. W. (1962). *Observation of a Markov process through a noisy channel.* Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Duri, S., Cole, A., Munson, J., & Christensen, J. (2001). An approach to providing a seamless end-user experience for location-aware applications. In *Proceedings of the First International Workshop on Mobile Commerce* (pp. 20–25). Rome.

Fano, A. E. (1998). SHOPPER'S EYE: Using location-based filtering for a shopping agent in the physical world. In K. P. Sycara & M. Wooldridge (Eds.), *AGENTS'98: Proceedings of the 2nd International Conference of Autonomous Agents* (pp. 416–421). New York: ACM Press.

Feng, Z., & Hansen, E. A. (2002). Symbolic heuristic search for factored markov decision processes. In *AAAI 2002: Proceedings of the Eighteenth National Conference on Artificial Intelligence* (pp. 455–460). Menlo Parc, CA, USA: AAAI Press.

Fikes, R. E., & Nilsson, H. J. (1971). STRIP: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *5*(2), 189–205.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). *PDDL—The planning domain definition language* (Tech. Rep. No. CVC TR-98-003/DCS). Yale Center for Computitational Vision and Control.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory and practice.* San Francisco: Elsevier, Morgan Kaufman.

Givan, R., Leach, S., & Dean, T. (1997). Bounded parameter Markov decision processes. In S. Steel & R. Alami (Eds.), *ECP'97: Proceedings of the Fourth European Conference on Planning* (pp. 234–246). Toulouse, France: Springer.

Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, *35*(12), 61–70.

Hansen, E. (1998). Solving POMDPs by searching in policy space. In G. F. Cooper & S. Moral (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference* (pp. 211–219). San Francisco: Morgan Kaufmann.

Hansen, E., & Feng, Z. (2000). Dynamic programming for POMDPs using a factored state representation. In S. Chien, S. Kambhampati, & C. A. Knoblock (Eds.), *AIPS 2000: Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling* (pp. 130–139). Breckenridge, Colorado.

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research, 13,* 33–94.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In G. F. Cooper & S. Moral (Eds.), *UAI'98: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (pp. 220–229). San Francisco: Morgan Kaufmann.

Herlocker, J. L., Konstan, J. A., & Riedl, J. (2000). Explaining collaborative filtering recommendations. In *Proceedings of the 2000 Conference on Computer-Supported Cooperative Work* (pp. 241–250). Philadelphia, PA.

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. In K. B. Laskey & H. Prade (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference* (pp. 279–288). San Francisco: Morgan Kaufmann.

Hoey, J., St.Aubin, R., Hu, A., & Boutilier, C. (2000). *Optimal and approximate stochastic planning using decision diagrams* (Tech. Rep. No. TR-2000-05). Department of Computer Science, University of British Columbia.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., & Rommelse, K. (1998). The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In G. F. Cooper & S. Moral (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Fourteenth Conference* (pp. 256–265). San Francisco: Morgan Kaufmann.

Howard, R. A. (1960). *Dynamic programming and Markov processes.* Cambridge, Massachusetts: The MIT Press.

Howard, R. A., & Matheson, J. E. (1984). Influence diagrams. In R. A. Howard & J. E. Matheson (Eds.), *Readings on the Principles and Applications of Decision Analysis* (Vol. 2, pp. 721–762). Strategic Decisions Group, Menlo Park, CA.

Jameson, A. (1989). But what will the listener think? Belief ascription and image maintenance in dialog. In A. Kobsa & W. Wahlster (Eds.), *User models in dialog systems* (pp. 255–312). Berlin: Springer.

Jameson, A. (1996). Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interaction, 5,* 193–251.

Jameson, A. (2002). Usability issues and methods for mobile multimodal systems. In *Proceedings of the ISCA tutorial and research workshop on multi-modal dialogue in mobile environments.* Kloster Irsee, Germany. (Summary of a keynote address)

Jameson, A. (2003). Adaptive interfaces and agents. In J. A. Jacko & A. Sears (Eds.), *Human-computer interaction handbook* (pp. 305–330). Mahwah, NJ: Erlbaum.

Jameson, A., & Buchholz, K. (1998). Einleitung zum Themenheft “Ressourcenadaptive kognitive Prozesse” [Introduction to the special issue on “Resource-Adaptive Cognitive Processes”]. *Kognitionswissenschaft*, *7*, 95–100.

Jameson, A., Großmann-Hutter, B., March, L., & Rummer, R. (2000). Creating an empirical basis for adaptation decisions. In H. Lieberman (Ed.), *IUI 2000: International Conference on Intelligent User Interfaces* (pp. 149–156). New York: ACM.

Jameson, A., Großmann-Hutter, B., March, L., Rummer, R., Bohnenberger, T., & Wittig, F. (2001). When actions have consequences: Empirically based decision making for intelligent user interfaces. *Knowledge-Based Systems*, *14*, 75–92.

Jameson, A., Sch¨afer, R., Simons, J., & Weis, T. (1995). Adaptive provision of evaluation-oriented information: Tasks and techniques. In C. S. Mellish (Ed.), *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1886–1893). San Mateo, CA: Morgan Kaufmann.

Jameson, A., Sch¨afer, R., Weis, T., Berthold, A., & Weyrath, T. (1999). Making systems sensitive to the user’s changing resource limitations. *Knowledge-Based Systems*, *12*, 413–425.

Jameson, A., & Schwarzkopf, E. (2002). Pros and cons of controllability: An empirical study. In P. De Bra, P. Brusilovsky, & R. Conejo (Eds.), *Adaptive Hypermedia and Adaptive Web-Based Systems: Proceedings of AH 2002* (pp. 193–202). Berlin: Springer.

Joachims, T., Freitag, D., & Mitchell, T. (1996). *WebWatcher: A tour guide for the World Wide Web* (Tech. Rep.). Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.

Joachims, T., Freitag, D., & Mitchell, T. (1997). WebWatcher: A tour guide for the World Wide Web. In M. E. Pollack (Ed.), *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (pp. 770–777). San Francisco, CA: Morgan Kaufmann.

John, B. E., & Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction*, *3*, 320–351.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kay, J. (2001). Learner control. *User Modeling and User-Adapted Interaction*, *11*, 111–127.

K¨ohler, J. (1994). *Reuse of plans in deductive planning systems.* Berlin: AKA Verlag.

Kjærulff, U. (1995). dHugin: A computational system for dynamic time-sliced Bayesian networks. *International Journal of Forecasting*, *11*, 89–111.

Kobsa, A., & Pohl, W. (1995). The user modeling shell system BGP-MS. *User Modeling and User Adapted Interaction*, *4*, 59–106.

Koehler, J. (1996). Planning from second principles. *Artificial Intelligence, 87*(1–2), 145–146.

Koehler, J. (1998). Planning under resource constraints. In H. Prade (Ed.), *ECAI'98: Proceedings of the 13th European Conference on Artificial Intelligence* (pp. 489–493). Chichester: John Wiley & Sons.

Koehler, J., & Schuster, K. (2000). Elevator control as a planning problem. In *AIPS 2000: Artificial Intelligence Planning Systems* (pp. 331–338).

Koller, D., & Parr, R. (2000). Policy iteration for factored MDPs. In C. Boutilier & M. Goldszmidt (Eds.), *UAI 2000: Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence* (pp. 326–334). Stanford, CA: Morgan Kaufmann Publishers.

Korf, R. E. (1985). Macro-operators: a weak method for learning. *Artificial Intelligence, 1985, 26,* 35–77.

Kray, C. (1995). *Situated interaction on spatial topics.* Berlin: AKA Verlag.

K¨upper, D., & Kobsa, A. (1999). User-tailored plan generation. In J. Kay (Ed.), *UM'99:, User Modeling: Proceedings of the Seventh International Conference* (pp. 45–54). Vienna: Springer Wien New York.

K¨upper, D., & Kobsa, A. (2001). User-tailored plan presentation. In M. Bauer, P. Gmytrasiewicz, & J. Vassileva (Eds.), *UM 2001: User Modeling: Proceedings of the Eighth International Conference* (pp. 243–246). Berlin: Springer.

Landauer, T. K. (1997). Behavioral research methods in human-computer interaction. In M. Helander, T. K. Landauer, & P. V. Prabhu (Eds.), *Handbook of human-computer interaction* (pp. 203–227). Amsterdam: North-Holland.

Lane, T., & Kaelbling, L. P. (2001). Toward hierarchical decomposition for planning in uncertain environments. In *Proceedings of the 2001 IJCAI Workshop on Planning under Uncertainty and Incomplete Information* (pp. 1–7). Seattle, WA: AAAI Press.

Lesh, N., Rich, C., & Sidner, C. L. (1999). Using plan recognition in human-computer collaboration. In J. Kay (Ed.), *UM'99, User modeling: Proceedings of the Seventh International Conference* (pp. 23–32). Vienna: Springer Wien New York.

Levin, E., Pieraccini, R., & Eckert, W. (2000). A stochastic model of human-machine interaction for learning dialogue strategies. *IEEE Transactions on Speech and Audio Processing, 8,* 11–23.

Lieberman, H. (1995). Letizia: An agent that assists Web browsing. In C. S. Mellish (Ed.), *IJCAI'95: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 924–929). Montreal, Quebec, Canada: Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Lindmark, K. (2000). *Interpreting symptoms of cognitive load and time pressure in manual input.* Unpublished master's thesis, Department of Computer Science, Saarland University, Germany.

Littman, M., Dean, T., & Kaelbling, L. (1995). On the complexity of solving Markov decision problems. In C. S. Mellish (Ed.), *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 394–402). San Mateo, CA: Morgan Kaufmann.

Littman, M. L. (1994). *The witness algorithm: Solving partially observable Markov decision processes* (Tech. Rep. No. CS-94-40). Department of Computer Science, Brown University.

Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1995). *Efficient dynamic-programming updates in partially observable Markov decision processes* (Tech. Rep. No. CS-95-19). Department of Computer Science, Brown University.

Lochbaum, K. E. (1998). A collaborative planning model of intentional structure. *Computational Linguistics*, *24*(4), 525–572.

Luenberger, D. G. (1979). *Introduction to dynamic systems: Theory, models, & applications.* Wiley.

Lusena, C., Goldsmith, J., & Mundhenk, M. (2001). Nonapproximability results for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, *14*, 83–103.

Madani, O., Hanks, S., & Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *AAAI'99: Proceedings of the 6th National Conference on Artificial Intelligence* (pp. 541–548). Menlo Park, Cal.: AAAI/MIT Press.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM*, *37*(7), 30–40.

Malaka, R., & Zipf, A. (2000). Deep Map - Challenging IT research in the framework of a tourist information system. In D. R. Fesenmaier, S. Klein, & D. Buhalis (Eds.), *ENTER2000: Proceedings of 7th. International Congress on Tourism and Communications* (pp. 15–27). Wien: Springer.

Maybury, M. T. (1991). Planning multimedia explanations using communicative acts. In *AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 61–66). Cambridge, Mass.: The MIT Press.

Maybury, M. T., & Wahlster, W. (Eds.). (1998). *Readings in intelligent user interfaces.* San Francisco, CA: Morgan Kaufmann.

Meuleau, N., Dearden, R., & Washington, R. (2004). Scaling up decision theoretic planning to planetary rover problems. In *AAAI 2004 Workshop on "Learning and Planning in Markov Processes–Advances and Challenges"at the Nineteenth National Conference on Artificial Intelligence.* San Jose, CA. (In press)

Meuleau, N., Hauskrecht, M., Kim, K.-E., Peshkin, L., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Solving very large weakly coupled Markov decision processes. In C. Rich & J. Mostow (Eds.), *AAAI'98: Proceedings of the 15th National Conference on Artificial Intelligence* (pp. 165–172). Menlo Park: AAAI Press.

Montemerlo, M., Pineau, J., Roy, N., Thrun, S., & Verma, V. (2002). Experiences with a mobile robotic guide for the elderly. In R. Dechter, R. Sutton, & M. Kearns (Eds.), *AAAI 2002: Proceedings of the Eighteenth National Conference on Artificial Intelligence* (pp. 587–592). Menlo Parc, CA, USA: AAAI Press.

Moore, A. W., Baird, L., & Kaelbling, L. P. (1999). Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. In D. Thomas (Ed.), *IJCAI'99: Proceedings of the 16th International Joint Conference on Artificial Intelligence* (pp. 1316–1323). S.F.: Morgan Kaufmann Publishers.

Moore, J. D., & Paris, C. L. (1993). Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics*, *19*, 651–694.

Murray, C., Van Lehn, K., & Mostow, J. (2001). A decision-theoretic approach for selecting tutorial discourse actions. In *Proceedings of NAACL Workshop on Adaptation in Dialogue Systems.* Pittsburgh, PA.

Ndiaye, A. (1998). *Rollenübernahme als Benutzermodellierungsmethode: Globale Antizipation in einem transmutierbaren Dialogsystem [Role taking as a method of user modeling: Global anticipation in a transmutable dialog system].* Unpublished doctoral dissertation, Department of Computer Science, Saarland University, Germany.

Neapolitan, R. E. (1990). *Probabilistic reasoning in expert systems: Theory and algorithms.* New York: Wiley.

Newcomb, E., Pashley, T., & Stasko, J. (2003). Mobile computing in the retail arena. In L. Terveen, D. Wixon, E. Comstock, & A. Sasse (Eds.), *Proceedings of ACM CHI 2003 Conference on Human Factors in Computing Systems* (Vol. 1, pp. 337–344). New York: ACM.

Newell, A., & Simon, H. A. (1963). GPS: A program that simulates human thought. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and Thought.* New York: McGraw-Hill.

Ng, A. Y., & Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In C. Boutilier & M. Goldszmidt (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference* (pp. 406–415). San Francisco: Morgan Kaufmann.

Nikovski, D., & Brand, M. (2003). Marginalizing out future passengers in group elevator control. In C. Meek & U. Kjærulff (Eds.), *UAI 2003: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence* (pp. 443–450). Morgan Kaufmann.

Nilim, A., & El Ghaoui, L. (2004). Robust solutions to markov decision problems. In *AAAI 2004 Workshop on "Learning and Planning in Markov Processes–Advances and Challenges"at the Nineteenth National Conference on Artificial Intelligence.* San Jose, CA. (In press)

Paek, T., & Horvitz, E. (2000). Conversation as action under uncertainty. In C. Boutilier & M. Goldszmidt (Eds.), *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference.* San Francisco: Morgan Kaufmann.

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of Operations Research*, *12*(3), 441–450.

Parr, R. (1998). Flexible decomposition algorithms for weakly coupled Markov decision problems. In G. F. Cooper & S. Moral (Eds.), *UAI'98: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence* (pp. 422–430). San Francisco: Morgan Kaufmann.

Pazzani, M. J. (2000). Representation of electronic mail filtering profiles: A user study. In H. Lieberman (Ed.), *IUI 2000: International Conference on Intelligent User Interfaces* (pp. 202–206). New York: ACM.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* San Mateo, CA: Morgan Kaufmann.

Penberthy, J. S., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In B. Nebel, C. Rich, & W. Swartout (Eds.), *KR'92: Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference* (pp. 103–114). San Mateo, California: Morgan Kaufmann.

Petrushin, V. A., & Sinitsa, K. M. (1993). Using probabilistic reasoning techniques for learner modeling. In P. Brna, S. Ohlsson, & H. Pain (Eds.), *Artificial Intelligence in Education: Proceedings of AI-ED 93* (pp. 418–425). Charlottesville, VA: Association for the Advancement of Computing in Education.

Pineau, J., Gordon, G., & Thrun, S. (2003). Policy-contingent abstraction for robust robot control. In C. Meek & U. Kjærulff (Eds.), *UAI 2003: Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence* (p. 477-484). Morgan Kaufmann.

Pineau, J., Roy, N., & Thrun, S. (2001). A hierarchical approach to POMDP planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learnin at the Eighteenth International Conference on Machine Learning.* Williams College, Massachusetts.

Plutowski, M. (2003). MDP solver for a class of location-based decisioning tasks. In *Proceedings of the Bayesian Modeling Applications Workshop at the Nineteenth Conference on Uncertainty in Artificial Intelligence.* Acapulco, Mexico.

Popp, H., & L¨odel, D. (1996). Fuzzy techniques and user modeling in sales assistants. *User Modeling and User-Adapted Interaction*, *5*, 349–370.

Pospischil, G., Umlauft, M., & Michlmayr, E. (2002). Designing LoL@, a mobile tourist guide for UMTS. In F. Paterno (Ed.), *Proceedings of the Fourth International Symposium on Human-Computer Interaction with Mobile Devices* (pp. 140–154). Berlin, Heidelberg, New York: Springer.

Poupart, P., & Boutillier, C. (2004). VDCBPI: An approximate scalable algorithm for large scale POMDPs. In *NIPS 2004: Advances in Neural Information Processing Systems.* Cambridge, MA: MIT Press. (to appear)

Pryor, L., & Collins, G. (1996). Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, *4*, 287–339.

Puterman, M. L. (1994). *Markov decision processes—discrete stochastic dynamic programming.* New York, NY: John Wiley & Sons, Inc.

Puterman, M. L., & Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision processes. *Management Science, 24*, 1127–1137.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*(2), 257–285.

Randell, C., & Muller, H. (2000). The shopping jacket: Wearable computing for the consumer. *Personal Ubiquitous Computing, 4*(4), 241–244.

Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM, 40*(3), 56–58.

Rich, C., & Sidner, C. L. (1998). COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction, 8*, 315–350.

Rich, C., Sidner, C. L., & Lesh, N. (2001). COLLAGEN: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*.

Rickel, J., Lesh, N., Rich, C., Sidner, C. L., & Gertner, A. (2002). Collaborative discourse theory as a foundation for tutorial dialogue. *Lecture Notes in Computer Science, 2363*, 542–551.

Riedl, M. O. (2001). A computational model and classification framework for social navigation. In *IUI 2001: International Conference on Intelligent User Interfaces* (pp. 137–144). New York: ACM.

Rist, T., André, E., & M¨uller, J. (1997). Adding animated presentation agents to the interface. In M. Johanna, E. Edmonds, & A. Puerta (Eds.), *IUI'97: Proceedings of the International Conference on Intelligent User Interfaces* (pp. 79–86). New York: ACM Press.

Roy, N., Pineau, J., & Thrun, S. (2000). Spoken dialogue management using probabilistic reasoning. In *ACL 2000: Proceedings of the Thirty-Eighth Meeting of the Association for Computational Linguistics.* Hong Kong.

Roy, N., & Thrun, S. (1999). Coastal navigation with mobile robots. In *NIPS'99: Advances in Neural Processing Systems* (Vol. 12, p. 1043-1049).

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Sacerdoti., E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5*(2), 115–135.

Sacerdoti, E. D. (1977). *A structure for plans and behavior.* Amsterdam and New York: Elsevier North-Holland.

Sawhney, N., & Schmandt, C. (2000). Nomadic Radio: Speech and audio interaction for contextual messaging in nomadic environments. *ACM Transactions on Computer-Human Interaction, 7*, 353–383.

Sedgewick, R. (2003). *Algorithms in Java* (Third ed.). Reading, MA, USA: Addison-Wesley. (Parts 1-4.)

Shafer, G. (1976). *A mathematical theory of evidence.* Princeton, New Jersey: Princeton University Press.

Shearin, S., & Lieberman, H. (2001). Intelligent profiling by example. In J. Lester (Ed.), *IUI 2001: International Conference on Intelligent User Interfaces* (pp. 145–151). New York: ACM.

Shekar, S., Nair, P., & Helal, A. S. (2003). iGrocer: A ubiquitous and pervasive smart grocery shopping system. In *Proceedings of the 2003 ACM Symposium on Applied Computing* (pp. 645–652). ACM Press.

Singh, M., Jain, A. K., & Singh, M. P. (1999). E-commerce over communicators: Challenges and solutions for user interfaces. In *Proceedings of the First ACM Conference on Electronic Commerce* (pp. 177–186). Denver.

Singh, S., & Cohn, D. (1998). How to dynamically merge Markov decision processes. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), *NIPS'97: Advances in Neural Information Processing Systems* (Vol. 10). The MIT Press.

Skaanning, C. (2000). A knowledge acquisition tool for bayesian-network troubleshooters. In *Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence* (pp. 549–557). San Francisco, CA: Morgan Kaufmann Publishers.

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, *21*, 1071–1088.

Smyth, B., & Cotter, P. (2002). The plight of the navigator: Solving the navigation problem for wireless portals. In P. De Bra, P. Brusilovsky, & R. Conejo (Eds.), *AH 2002: Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web Systems* (pp. 328–337). Malaga, Spain: Springer.

Sondik, E. (1971). *The optimal control of partially observable Markov processes.* Unpublished doctoral dissertation, Stanford University.

St-Aubin, R., Hoey, J., & Boutilier, C. (2001). APRICODD: Approximate policy construction using decision diagrams. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *NIPS 2002: Advances in Neural Information Processing Systems* (pp. 1089–1095). MIT Press.

Sumi, Y., & Mase, K. (2001). Digital assistant for supporting conference participants: An attempt to combine mobile, ubiquitous and Web computing. In G. D. Abowd, B. Brumitt, & S. Shafer (Eds.), *UbiComp 2001: Ubiquitous Computing* (pp. 156–175). Berlin: Springer.

Svensson, M., H¨o¨ok, K., Laaksolahti, J., & Waern, A. (2001). Social navigation of food recipes. In J. A. Jacko, A. Sears, M. Beaudouin-Lafon, & R. J. Jacob (Eds.), *Human Factors in Computing Systems: CHI 2001 Conference Proceedings* (pp. 341–348). New York: ACM.

Thorndyke, P. W., & Hayes-Roth, B. (1982). Differences in spatial knowledge acquired from maps and navigation. *Cognitive Psychology*, *14*, 560–589.

Thrun, S. (2000). Monte Carlo POMDPs. In S. A. Solla, T. K. Leen, & K.-R. M¨uller (Eds.), *NIPS 2000: Advances in Neural Information Processing* (p. 1064-1070). The MIT Press.

Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, *7*(1), 81-120.

von Winterfeldt, D., & Edwards, W. (1986). *Decision analysis and behavioral research.* Cambridge, England: Cambridge University Press.

Wahlster, W. (2002). SmartKom: Fusion and fission of speech, gestures and facial expressions. In *Proceedings of the First International Workshop on Man-Machine Symbiotic Systems* (pp. 213–225). Kyoto.

Wahlster, W., André, E., Finkler, W., Profitlich, H.-J., & Rist, T. (1993). Plan-based integration of natural language and graphics generation. *Artificial Intelligence*, *63*, 387–427.

Wahlster, W., Baus, J., Kray, C., & Kr¨uger, A. (2001). REAL: Ein ressourcenadaptierendes mobiles Navigationssystem. *Informatik Forschung und Entwicklung*, *16*(4), 233-241.

Wahlster, W., Reithinger, N., & Blocher, A. (2001). SmartKom: Multimodal communication with a life-like character. In *Eurospeech 2001: Proceedings of the 7th European Conference on Speech Communication and Technology* (pp. 1547–1550). Aalborg, Denmark.

Wahlster, W., & Tack, W. (1997). SFB 378: Ressourcenadaptive Kognitive Prozesse. In M. Jarke (Ed.), *Informatik'97: Informatik als Innovationsmotor* (pp. 51–57). Berlin: Springer.

Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, *12*, 387–416.

Walker, M. A., J., D. L., Kamm, C. A., & Abella, A. (1997). PARADISE: A framework for evaluating spoken dialogue agents. In P. R. Cohen & W. Wahlster (Eds.), *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics* (pp. 271–280). Somerset, New Jersey: Association for Computational Linguistics.

Wexelblat, A., & Maes, P. (1997). *Issues for software agent UI.* (Unpublished manuscript, available from http://wex.www.media.mit.edu/people/wex/)

Wittig, F. (2003). *Maschinelles Lernen Bayes'scher Netze für benutzeradaptive Systeme.* Berlin: Aka Verlag.

Yang, Q. (1997). *Intelligent planning: A decomposition and abstraction based approach to classical planning.* Berlin: Springer-Verlag.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.

Zadeh, L. A. (1968). Fuzzy algorithms. *Information and Control*, *12*(2), 94–102.

Zhang, N., & Zhang, W. (2001). Speeding up the convergence of value iteration in partially observable Markov decision processes. *Artificial Intelligence Research*, *14*, 29–51.

Zhang, N. L., & Liu, W. (1996). *Planning in stochastic domains: Problem characteristics and approximation* (Tech. Rep. No. HKUST-CS96-31). Department of Computer Science, Hong Kong University of Science and Technology.

Zhou, R., & Hansen, E. A. (2001). An improved grid-based approximation algorithm for POMDPs. In B. Nebel (Ed.), *IJCAI 2001: Proceedings of the Seventeenth International Conference on Artificial Intelligence* (pp. 707–716). San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, *17*(3), 73–83.

Index