Universität des
Saarlandes

Resource-Aware Plan Recognition in Instrumented Environments

Michael Schneider

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

24. Februar 2010

Dekan:

Prof. Dr. Joachim Weickert

Vorsitzender des Prüfungsausschusses:

Prof. Dr. Philipp Slusallek

Berichterstatter:

Prof. Dr. Dr. h.c. mult. Wolfgang Wahlster

Prof. Dr. Antonio Krüger

Akademischer Beisitzer:

Dr. Patrick Gebhard

Tag des Kolloquiums:

22. Februar 2010

Eidesstattliche Versicherung

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

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

Saarbrücken, den 24. Februar 2010

Danksagung

Diese Arbeit entstand im Rahmen der vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Projekte SharedLife und SemProM am Deutschen Forschungszentrum für künstliche Intelligenz (DFKI).

Zu der Entstehung dieser Arbeit haben viele Personen durch ihre direkte und indirekte Unterstützung beigetragen. Ihnen gebührt mein Dank!

Ich danke meinem Doktorvater Prof. Wolfgang Wahlster für die Möglichkeit, in seiner Gruppe am Deutschen Forschungszentrum für künstliche Intelligenz mitarbeiten und meine Doktorarbeit anfertigen zu können. Vor allem in den letzten Phasen meiner Arbeit unterstütze er mich mit konstruktiven Diskussionen in ausgesprochen angenehmer Atmosphäre und lieferte wertvolle Denkanstößen zur Verbesserung meiner Arbeit. Prof. Antonio Krüger danke ich für seine Bereitschaft, das Zweitgutachten für diese Arbeit zu verfassen.

Während meiner Zeit am DFKI hatte und habe ich das Glück, mit vielen netten Kollegen zusammenarbeiten zu dürfen. Ihnen danke ich für die angenehme Arbeitsatmosphäre, für interessante und anregende Diskussionen, insbesondere aber auch für das entgegengebrachte Verständnis und Ihre Rücksichtnahme während des Niederschreibens und der heißen Schlussphase meiner Arbeit. Ohne die überlassenen Freiräume wären die zugrundeliegende Forschung sowie das Verfassen der Arbeit in dieser Form kaum möglich gewesen.

Für das Probelesen von Kapiteln danke ich insbesondere Jörg Baus, Michael Kipp und Alexander Kröner. Für das Teilen meines Leids in schwierigen Phasen danke ich allen ehemaligen und aktuellen Doktoranden und Mitarbeitern am Lehrstuhl Prof. Wahlster und seiner Arbeitsgruppe am DFKI.

Ein besonderer Dank gilt zum Schluss meinen Eltern, Conny, meinem Bruder und meinen Freunden, die mich – aus der Nähe und auch aus der Ferne – stets nach Kräften unterstützt und gefördert haben. Ohne ihre moralische und praktische Unterstützung hätte diese Arbeit nicht entstehen können!

Saarbrücken, im Februar 2010

Michael Schneider

Zusammenfassung

Diese Arbeit behandelt das Problem der *Planerkennung* in *instrumentierten Umgebungen*. Ziel ist dabei das Erschließen der *Pläne des Nutzers* anhand der *Beobachtung seiner Handlungen*. In instrumentierten Umgebungen erfolgt diese Beobachtung über *physische Sensoren*. Dies wirft spezifische Probleme auf, von denen zwei in dieser Arbeit näher betrachtet werden:

- Physische Sensoren beobachten in der Regel *Zustände* anstelle direkter *Nutzeraktionen*. Klassische Planerkennungsverfahren basieren jedoch auf der Beobachtung von Aktionen, was bisher eine *aufwendige und fehlerträchtige Ableitung von Aktionen* aus Zustandsbeobachtungen notwendig macht.

- Aufgrund *beschränkter Resourcen* der Umgebung ist es oft nicht möglich alle Sensoren gleichzeitig zu aktivieren. Aktuelle Planerkennungsverfahren bieten keine Möglichkeit, die Umgebung bei der *Auswahl einer relevanten Teilmenge von Sensoren* zu unterstützen.

Diese Arbeit beschreibt einen zweistufigen Ansatz zur Lösung der genannten Probleme. Zunächst wird ein *DBN-basiertes Planerkennungsverfahren* vorgestellt, das Zustandswissen explizit repräsentiert und in Schlussfolgerungen berücksichtigt. Dieses Verfahren bildet die Basis für ein *POMDP-basiertes Nutzenmodell* für Beobachtungsquellen, das für den Zweck der Sensorauswahl genutzt werden kann.

Des Weiteren wird ein *Toolkit* zur Realisierung von Planerkennungs- und Sensorauswahlfunktionen vorgestellt sowie die *Gültigkeit und Performanz* der vorgestellten Modelle in einer *empirischen Studie* evaluiert.

Summary

This thesis addresses the problem of *plan recognition* in *instrumented environments*, which is to infer an *agent's plans* by *observing its behavior*. In instrumented environments such observations are made by *physical sensors*. This introduces specific challenges, of which the following two are considered in this thesis:

- Physical sensors often observe *state information* instead of *actions*. As classical plan recognition approaches usually can only deal with action observations, this requires a *cumbersome and error-prone inference of executed actions* from observed states.
- Due to *limited physical resources* of the environment it is often not possible to run all sensors at the same time, thus *sensor selection* techniques have to be applied. Current plan recognition approaches are not able to support the environment in *selecting relevant subsets of sensors*.

This thesis proposes a two-stage approach to solve the problems described above. Firstly, a *DBN-based plan recognition approach* is presented which allows for the explicit representation and consideration of state knowledge. Secondly, a *POMDP-based utility model* for observation sources is presented which can be used with generic utility-based sensor selection algorithms.

Further contributions include the presentation of a *software toolkit* that realizes plan recognition and sensor selection in instrumented environments, and an *empirical evaluation* of the *validity and performance* of the proposed models.

Contents

The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' but 'That's funny...'

Isaac Asimov (1920–1992)

1 Introduction

1.1 Motivation and Goals

Plan Recognition is the process of observing an agent's behavior in order to *learn about its plans*, to *infer its goals*, and to *predict its future actions*. In this context, the term agent can refer to any autonomous entity, including human users, animals, and even autonomous technical systems like robots or software programs.

With their multitude of sensors, *instrumented environments* provide a powerful infrastructure for the *seamless observation* of agents and thus the application of plan recognition techniques in real-world scenarios. In return, instrumented environments can utilize plan recognition information in order to provide *proactive assistance*, for instance by *anticipating the agent's needs*, *adapting the environment* to upcoming tasks, *providing context-dependent information* which helps the agent reaching its goals more efficiently, the *execution of actions on behalf of the agent*, or even by *actively preventing* the agent from executing undesired or harmful actions and plans.

The utility of plan recognition in instrumented environments is illustrated by the following example from the domain of *Ambient Assisted Living*[1]:

> Imagine an instrumented kitchen environment that can observe the use of ingredients, utensils, and kitchen appliances via unobtrusively installed sensors. The environment furthermore is equipped with a screen and speaker system which can be used to provide its user with information. Furthermore, the major functions of the installed kitchen appliances can be remotely controlled. From the user's perspective, the kitchen is used like any regular kitchen.

[1] Ambient Assisted Living (AAL) includes methods and technologies which unobtrusively support elderly or handicapped users in their daily live by adapting themselves to the users' needs and capabilities.

> Now imagine that John – an elderly user with cognitive impairments due to dementia – starts preparing a cake in this kitchen. Through the use of plan recognition the kitchen environment autonomously infers from the used ingredients and executed preparation steps that John is most probably preparing a hazelnut cake. Based on this knowledge, the system guides John through the required steps by proactively presenting context-dependent preparation instructions. The kitchen constantly monitors if John deviates from the recipe, reminds him on missed ingredients, or warns John in case that he intends to add required ingredients for the second time. The system further reminds John on the right time to remove the cake from the oven, and turns of the oven on behalf of John if he forgets to do so after baking is finished.

Classical plan recognition approaches (cf. [Car01, AA07]) have mainly been applied in virtual application domains like operating systems or computer games, where actions of observed agents can often be directly observed via appropriate software probes. These probes usually have full access to the virtual system's internal data structures and message queues, from which they can easily and cheaply read all necessary information. In contrast to plan recognition in virtual environments, plan recognition in real-world environments has to rely on physical sensors to observe an agent and its behavior. This introduces some *specific challenges* for plan recognition in such environments, which originate from particular properties of the applied sensing techniques:

A first difference is, that due to their principle of operation physical sensors usually observe (partial) information about the *state of the environment*, but generally *do not allow for the direct observation of executed actions*. An example is an RFID sensor (see subsection 2.3.2), which observes that a pack of flour is located on the kitchen's countertop, but cannot tell whether the pack was purposefully put there by the observed agent, or accidentally fell out of the cupboard. A second example is a light sensor, which observes that the light level dropped, but alone cannot tell whether this was caused by an agent turning off the light, or by some cloud passing by the window. Both examples show that in the general case it is difficult or even impossible to reliably infer the execution of actions from the observation of states, especially if more complex actions than the moving of objects or the switching of lights should be observed. The unavailability of direct action observations is a critical problem, as most existing plan recognition approaches reason about sequences of observed symbolic actions, and cannot directly deal with observations regarding the environment's state.

Another difference between plan recognition in virtual and physical environments are the costs that are associated with the execution of sensing actions. Physical sensing generally is associated with higher costs than sensing in virtual environments, in particular if *operating physical sensors consumes limited and thus valuable resources*. An example are battery-driven nodes of a wireless sensor network (see subsection 2.3.1). Every sensing, data processing, and communication act draws electrical energy from the batteries. If constantly operated, the batteries of nodes might be quickly drained, which results in an outage of nodes and requires the user or some technician to replace the batteries. Another example is communication bandwidth. As all network nodes use the same frequency for radio communication, only a limited amount of nodes can simultaneously communicate without jamming the communication channel.

These examples show, that if a huge number of sensors is available to observe an agent in an environment, it is often reasonable or required to *restrict the set of sensors that are active* at the same point in time due to economical or physical resource constraints. In such a case, a subset of sensors is sent into sleep mode or even is temporarily deactivated to preserve resources and to avoid sensing costs. Ideally, one would then prefer to *deactivate sensors which are "less" important* in the current situation in favor of sensors which are more "important". To the best of our knowledge there currently exists *no approach that allows for judging the importance or usefulness of observation information sources* with respect to an underlying plan recognition application. This currently prevents the use of sensor selection techniques in real-world plan recognition system.

The above-mentioned challenges associated with plan recognition in instrumented environments provide the main motivation for the research underlying this thesis. They are addressed as follows: The issue of *State Observations* is solved by proposing a plan recognition approach which explicitly takes into account state observations in its plan library and reasoning processes. The issue of *Resource-Constrained Sensing* is addressed by the presentation of a formal utility model which uses a decision-theoretic approach to judge the utility of observation information with respect to plan recognition. This utility model can then be used to perform sensor selection in real-world plan recognition applications.

Further goals of this thesis include the development of a software toolset which integrates sensor selection for plan recognition, plan recognition, and decision-making about support that should be provided based on the resulting plan hypothesis, the implementation of an instrumented environment as a testbed for the application of plan recognition, and an empirical evaluation of the proposed utility model in sensor selection applications.

1.2 Integration Into Related Research Topics

The work on resource-aware plan recognition in instrumented environments that is presented in this thesis lies in the intersection of three main research topics (see Figure 1.1):

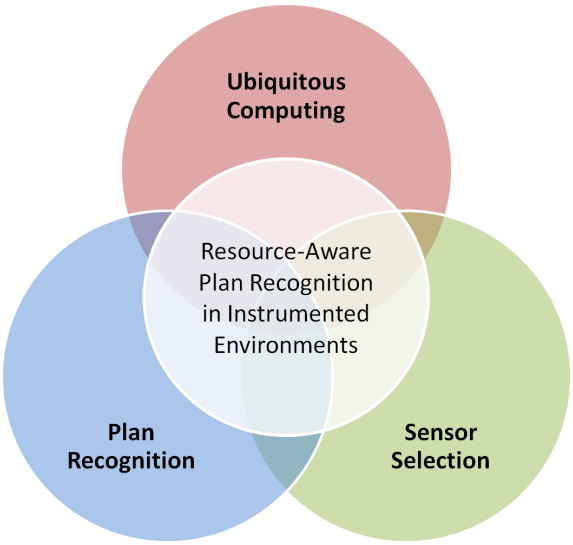


Figure 1.1: The work presented in this thesis lies in the intersection of the topics *Plan Recognition*, *Sensor Selection*, and *Ubiquitous Computing*.

- **Ubiquitous Computing:** The research topic of ubiquitous computing follows the vision, that *inexpensive, small, and networked devices* are distributed throughout our physical environment and are integrated into many everyday objects. These devices *dynamically cooperate* in order to provide the human user with *novel kinds of ambient services* [Wei91]. As opposed to the desktop paradigm, in which a single user consciously uses a single device for a specialized purpose, someone "using" ubiquitous computing engages *many computational devices and systems simultaneously*, in the course of ordinary activities, and may not necessarily even be aware of them. Ubiquitous computing and plan recognition complement each other perfectly: On the one hand, ubiquitous computing provides a *powerful infrastructure* for the seamless observation of the user's actions in an environment. On the other hand, plan recognition can provide valuable information about the user's goals and anticipated upcoming actions, which can be exploited by *ambient services* to adapt themselves to the user's needs

and to offer proactive assistance. On the downside, many of the devices used in ubiquitous computing environments face severe resource restrictions like weak power supply or limited wireless communication bandwidth, which increases the need for resource-aware sensor selection. A general introduction to ubiquitous computing environments is given in chapter 2. Concrete examples of such environments are presented in section 5.1.

- **Plan Recognition:** The research topic of plan recognition is a branch of *artificial intelligence* which is concerned with *identifying an autonomous agent's plans and goals* by observing its behavior. Such knowledge allows intelligent support systems to *provide proactive assistance*. This work is motivated by some of the special challenges that arise from the application of plan recognition in physical environments. One of these challenges is the problem of *limited sensing resources*, which might require to apply sensor selection in order to decide on a subset of sensors to use. Plan recognition provides an excellent opportunity for the application of sensor selection techniques, as the resulting plan hypothesis can be used to predict the agent's next actions and likely follow-up states of the environment. This information can be exploited to judge the utility of particular sensors in the near future, and thus helps sensor selection to decide on the "most promising" subset of sensors. A general introduction to plan recognition is given in chapter 3. A detailed discussion of related work in the field of plan recognition is provided in section 5.2.

- **Sensor Selection:** The research topic of sensor selection considers situations, where for *technical or economical reasons* only a subset of the sensors in an environment could or should be run at the same point in time. The problem of sensor selection then is to choose the *subset of sensors which promise to provide the most "valuable" information* with respect to some underlying application, while at the same time meeting some secondary criterion, like a *maximum upper bound on the total amount of consumed resources*. Existing sensor selection approaches are either application-specific or generic. In the generic case, sensor selection utilizes some external, application-specific measure to judge the quality of a selected subset of sensors. One of the goals of this thesis is to develop such a utility measure for observation information in plan recognition applications. Examples of typically applied sensing techniques and their inherent resource restrictions is given in section 2.3. Related work in the field of sensor selection is discussed in section 5.3.

1.3 Main Research Questions

The primary aim of the work presented in this thesis is to improve the applicability of plan recognition in instrumented environments by proposing a plan recognition approach which naturally allows for representing and processing of observation information from physical sensors, and which supports an eventually applied sensor selection algorithm by providing a utility model that allows for predicting the value of observation information with respect to the underlying plan recognition process. In order to reach this aim, the research which is presented in this thesis was guided by the following seven main research questions:

1. **Representation of State Information:** *What is a suitable plan selection and execution model that explicitly considers state information?*

   A formal model of the observed agent's plan selection and execution process (often called the agent's plan library) usually provides the foundation for a plan recognition system's reasoning. In order to develop a plan recognition system which naturally "understands" state observations, the applied plan selection and execution model should be expressed in a way that allows for the representation of state information in the plan library.

2. **State-Aware Plan Recognition:** *How can this model be used to perform state-aware plan recognition?*

   Technically, a plan recognition system has to infer (and update) plan hypotheses from observed information based on a known plan library. Given a concrete plan selection and execution model according to the requirements specified in the answer to research question 1, we need to know how to perform such inference on this model if we receive a set of state observations.

3. **Measure of Utility:** *What is a good measure of the utility of observation information with respect to the plan recognition problem?*

   In order to perform sensor selection in real-world environments we need to judge the importance of observation information with respect to its value for the application which utilizes the information. In the case of this thesis, this application is the plan recognition system. To compare the importance of different information items, a numerical measure for observation information utility in plan recognition applications has to be found.

4. **Influencing Factors:** *What are the factors that influence the concrete amount of observation information utility?*

   Once a suitable measure for information utility in plan recognition applications has been identified, the next step is to create a formal model which allows for the estimation of expected utility values for possible upcoming observations. In order to develop such a model, we have to learn about the factors which influence the concrete amount of observation utility.

5. **Formal Framework:** *Which formal framework can be used to model the relationship between these factors and the utility of information?*

   The utility model to be developed should be grounded on a well-defined theoretical framework, which is computationally feasible yet powerful enough to account for all factors which influence the utility of observation information that have been identified in the answer to research question 4. The identified relationships then have to be represented using the chosen theoretical framework.

6. **Estimation of Utility:** *How can this framework be used to estimate the expected utility of future observation information?*

   In order to solve the problem of sensor selection for plan recognition, the utility model that has been developed as a result of research question 5 has to be evaluated at runtime to estimate the utility value of expected observation information. This process has to be repeated several times during the observed agent's plan execution for different candidate sets of sensors. Thus, runtime performance of the estimation algorithm is a critical factor with respect to real-time applications.

7. **Performance:** *How does the proposed decision-theoretic utility model perform in sensor selection applications?*

   The performance of the overall system depends on the quality of the utility model's estimates. The better these estimates are, the better is the relevance of selected sensors for plan recognition, the better are the resulting plan hypothesis, and finally the better the provided proactive support can be. Thus, it is important to evaluate the suitability and performance of the proposed utility model for sensor selection in plan recognition applications, which was taken as motivation for the presented utility model for observation information.

1.4 Chapter Outline

This thesis is organized into three main parts: An *introduction and background* part, a *theory* part, and an *application and evaluation* part. The three parts are further subdivided into eleven chapters, which are described in the following. The overall resulting structure of this thesis is shown in Figure 1.2.

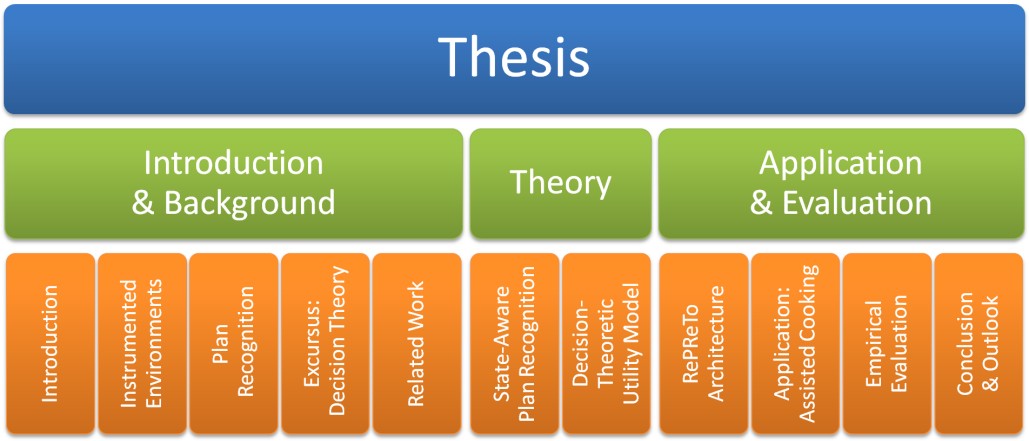


Figure 1.2: This thesis is organized into three main parts, which are further subdivided into eleven chapters.

Introduction and Background

The introduction and background part comprises five chapters. Chapter 1 provides the *motivation* for the conducted research, describes the integration into *related research topics*, introduces the seven *main research questions* that are addressed in this thesis, and provides an overview about the structure of this thesis. Chapter 2 gives an *introduction to instrumented environments*. After motivating the general idea, a formal definition of instrumented environments is presented. Next, the issue of *resource-constrained sensing* in instrumented environments is introduced, and the general problem of *sensor selection* is presented and defined. Chapter 3 provides an *introduction to plan recognition*. The general concept and algorithm are presented, and a formal definition of plan recognition systems is provided. We also give a definition of *instrumented environments with plan recognition support*, which constitute the main application domain for the work presented in this thesis. Chapter 4 contains a discourse on *reasoning under uncertainty* and *decision*

theory, which provides the theoretical foundation for the plan recognition approach and observation information utility model that are proposed in the theoretical part of this thesis. Chapter 5 concludes the introductory part with an overview about *related work* in the areas of instrumented environments, plan recognition, sensor selection, and reasoning about knowledge models.

Theory

The second part of this thesis is split into two chapters and introduces the *theoretical models and algorithms* that we developed to address the problems that have been identified in the introductory part. Chapter 6 presents our *state-aware plan recognition approach*, which provides the theoretical foundation for the rest of the work presented in this thesis. We start by presenting the general idea of the approach, which is based on an extended probabilistic finite automaton model of the observed agent's *plan selection and execution process* (also called the agent's plan library). After we have formally introduced this model, we describe our *sensor model*, which links abstract automaton states to partial states of the environment. We then describe how to derive a dynamic Bayesian network (DBN) from both models that allows to perform plan recognition in instrumented environments, and how to use this network to *predict future actions and anticipated goals*. Chapter 7 introduces a *utility model for observation information in plan recognition applications*. The chapter starts with a discussion of the nature of utility in plan recognition applications, and then develops a decision-theoretic utility measure based on the added value that a higher-order support system realizes through the use of plan recognition information. We formally introduce the underlying *support decision problem*, and show how to solve this problem by deriving a partially observable Markov decision problem (POMDP) from the plan library and an additional support model.

Application and Evaluation

The final part of this thesis comprises four chapters: Chapter 8 addresses the *practical application* of the proposed theoretical models and algorithms in real-world systems. The chapter introduces the *Resource-aware Plan Recognition Tool Set* (RePReTo), which describes an integrated architecture for a tool set that supports plan recognition, sensor selection, and proactive decision-making about the provision of support services in instrumented environments. We present the required knowledge models and software components, and describe the existing causal and temporal dependencies. We further discuss the problem of *knowledge model acquisition* in practical ap-

plication domains, and present some useful *design patterns for plan libraries*. Chapter 9 describes the implementation of an existing *smart kitchen environment*, which was developed during the research for this thesis as a testbed for the application of plan recognition techniques in instrumented environments. We present the technical setup of the kitchen environment, and introduce the *Semantic Cookbook* application which uses plan recognition to support a human user in the preparation of food. Chapter 10 describes the setup and results of an *empirical evaluation study* that was conducted using REPRETO to evaluate the suitability and performance of the proposed utility model for the purpose of sensor selection in plan recognition applications. Chapter 11 concludes this thesis with a *revision of the main research questions*, a discussion of the *major scientific contributions* of this thesis, and an *outlook on possible directions for future research* on the topic of resource-aware plan recognition in instrumented environments.

2 Instrumented Environments

This chapter introduces *instrumented environments*, which provide the application domain for the work presented in this thesis. We introduce the *general concept* of instrumented environments (see section 2.1) and provide a *formal definition* (see section 2.2). We discuss the problem of *resource-constraint sensing* and give examples of typical sensing technologies and their inherent resource restrictions (see section 2.3). These resource restrictions and the resulting need to perform *sensor selection* (see section 2.4) motivate the development of a utility model for sensor information (see chapter 7).

2.1 General Concept

Instrumented environments are *parts of the real-world* that are equipped with a *ubiquitous computing infrastructure* in order to realize added-value services for the environment's owner and/or user. This infrastructure typically comprises different *sensors* which allow the environment to perceive the current context, including information about the physical state of the environment or the actions of the user. The environment can interact with the user and/or adapt its physical state through a set of *actuators* like information displays (including visual, auditory, and tactile displays), or physical actuators and remotely controlled appliances. In addition, environments can contain intelligent objects (so-called *smart items*), which are equipped with autonomous intelligence, sensors, and actuators. In addition, a communication infrastructure allows these items to (wirelessly) communicate and cooperate. The physical elements of the environment are usually augmented with a virtual software and data layer, which links the physical environment to digital information sources and applications. The overall infrastructure is utilized by one or more *support applications*, which are part of the environment's virtual layer and utilize this infrastructure to implement a set of *added-value services*, which assist the user of the environment in performing its tasks.

An example for an instrumented environment is a smart kitchen environment, which is equipped with sensors that allow observing the user's cooking process. Possible sensors might include the detection of used ingredients and kitchen tools, or the user's interactions with electric kitchen appliances. Actuators might for instance include a screen and speakers that allow for presenting information or notifying the user, as well as the remote control of selected functions of the installed kitchen appliances through software applications. Support applications then could assist the user of the kitchen in the preparation of food, e.g. by teaching new recipes, presenting context-aware preparation information, warning on critical mistakes, or by presenting tips for improving the preparation process. The concrete realization of such an environment is presented in chapter 9.

In comparison to traditional computer systems, which require the user to explicitly interact with applications through generic interfaces like keyboard, mouse, or touch screen, instrumented environments allow for a more natural way of *implicit interaction* through the environment itself. Ideally, an environment can infer all information which is required to provide the offered services from sensor data and background knowledge, and does not require any direct input from the user. This approach has numerous advantages:

- **Lowered Inhibition**: Many people feel uncomfortable using computer systems because they are overstrained by the complexity of today's systems. This especially holds for elderly or technically inexperienced users. Instrumented environments try to avoid direct and explicit interaction between the computer system and the user wherever possible, and thus lower the inhibition to use computer systems. Instead of directly operating a computer, users can interact with an environment and the contained objects in a familiar and intuitive way, and nevertheless benefit from the assistance that is provided through the computer applications that are interwoven with the physical environment.

- **Flattened Learning Curve**: Most untrained people can operate a screwdriver without studying a user manual. Its physical properties like shape and used materials provide so-called *affordances*, which allow users to conclude the screwdriver's intended use. This principle can be applied to instrumented environments in order to ease the user's familiarization with contained applications: If functions of the environment are related to established physical, conceptual, or cultural concepts, this allows users to (often unconsciously) exploit the numerous experiences collected in the physical world over time in order to accelerate the process of learning.

- **Reduced Cognitive Load**: Performing real-world tasks with the support of traditional computer systems often causes additional cognitive load for the user: Besides concentrating on their primary tasks, users additionally have to deal with the burden of handling the computer application. By relocating the interaction with the computer system into the physical environment, instrumented environments allow for the seamless integration of the computer support system into the natural work flow, and thus can significantly lower the overall cognitive load. The actual technology diffuses into the background of the user's awareness and thus becomes "calm technology" according to Weiser's vision [Wei91] of ubiquitous computing.

- **Increased Efficiency**: As the interaction with an instrumented environment is typically driven by the user's actions in the physical world, the interaction between the user and available support applications is not constrained by predefined dialog boxes or static graphical user interfaces. Instead, the user can utilize the physical environment to its full extent, while the system tries its best to support the user in doing so. What on the one hand is a challenge regarding the design and implementation of instrumented environments, on the other hand allows the user to efficiently pursue her tasks without following a predefined path dictated by some computer system.

Realizing all these theoretical advantages requires careful design and implementation of the actual instrumented environments and support applications, which is not at all a trivial task. Resulting environments succeed variably well in reaching these goals, and the design and implementation of instrumented environments still is a very young and active research area.

2.2 Formal Definition

A formal definition of instrumented environments was introduced by us in [Sch03]. The definition accounts for the four central elements that characterize each instrumented environment: Its being part of the physical world (represented by the environments *spatial extension*), its intended use or *purpose*, its virtual software and *data layer*, and its *instrumentation* with information technology (in particular sensors and actuators). Based on these elements, Definition 2.2.1 introduces seven conditions which define an instrumented environment (see Figure 2.1 for a graphical illustration of the resulting dependencies).

Definition 2.2.1. An **instrumented environment** is a quadruple (E, P, D, I), where

1. E is the *spatial extension* of the environment in the physical world.
2. P it the *purpose* of the environment.
3. D is the set of digital items like data and applications which comprise the *virtual layer* of the environment (elements of D have no physical appearance).
4. I is the set of physical items like sensors, actuators, and communication infrastructure components which comprise the environment's *instrumentation* (elements of I always have a physical appearance).
5. Elements from D and I *together* support the *realization* of P (denoted $D \cup I \rightsquigarrow P$). "Together" means, that there exist no two disjoint subsets of $(D \cup I)$, which on their own allow to provide assistance: $\nexists X, Y \subset (D \cup I) : (D \cap I = \emptyset) \wedge (X \rightsquigarrow P) \wedge (Y \rightsquigarrow P)$.
6. Every element from the set $(D \cup I)$ contributes to the realization of P, thus $\forall x \in (D \cup I), \exists Y \in (D \cup I) \setminus \{x\} : (Y \not\rightsquigarrow P) \wedge (Y \cup \{x\} \rightsquigarrow P)$.
7. E is the smallest possible spatial extension in which P can be realized with the support of $(D \cup I)$. This implies $\forall i \in I : i \in E$.

The first four conditions formally introduce the components spatial extension (E), purpose (P), data layer (D), and instrumentation (I) as described above. The next three conditions help distinguishing instrumented environments from other entities with similar structure but different meaning. In particular, they restrict E, D, and I to the necessary minimums which still realize purpose P and have to be understood as follows:

The fifth condition serves two goals. On the one hand, it requires that virtual layer D and instrumentation I have to be sufficient to realize the intended purpose P. On the other hand, it defines that all components of the virtual layer and the instrumentation have to work together in order to realize that purpose. This prevents for instance, that a room with multiple non-networked computers is perceived as an instrumented environment for the manipulation of computer files. The sixth condition restricts the set of

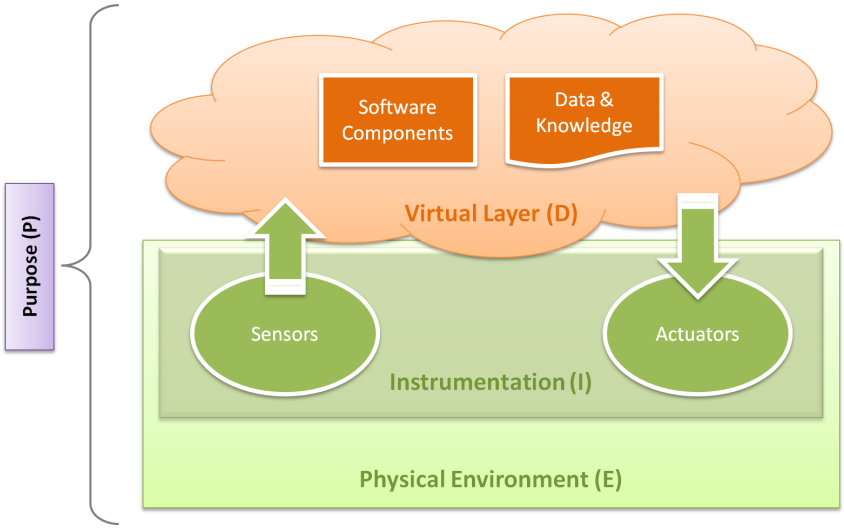


Figure 2.1: Relationship of components which comprise an instrumented environment (E, P, D, I).

items which form virtual layer D and instrumentation I to elements, which actually are required to realize P. Assume that the user in an instrumented environment owns a mobile phone. Although this phone in principle could act as a sensor and/or actuator, it is not part of the instrumented environment as long as it is not utilized by the virtual layer to help realizing P. Condition seven finally limits the spatial extension E of the environment. The value of E might be any spatial concept which describes a discrete part of the real world, like a building, a room, or even a single shelf. Condition seven demands that this concept is of minimal spatial expansion, thus for instance preventing a house from being perceived as an instrumented environment only because it contains an instrumented kitchen.

The above definition also covers *hierarchical instrumented environments*. These are instrumented environments that are composed of several "smaller" instrumented environments which cooperate to provide additional services that span several of the contained environments. As an example, multiple smart home environments if connected can constitute a smart city environment which provides services like energy management or pervasive user adaptation across multiple sites and buildings. Other elements that might be part of such a smart city environment are smart shopping environments and intelligent traffic systems like instrumented roads or smart cars.

2.3 Resource-Constrained Sensing

Instrumented environments have to rely on physical sensors to observe and support their users. In this section we discuss issues related to sensing in real-world environments with a special emphasis on the costs that are associated with different sensing approaches. As the *cost of sensing* we understand the sum of resources like electrical power, communication bandwidth, CPU time, memory, etc., that are "consumed" by a sensor if it is activated.

Usually, the amount of resources that is available in a physical environment is limited due to technical reasons and laws of physics. If the available resources are exhausted, the system can no longer operate as expected, which might result in local or global malfunctions and in the worst case may lead to unpredictable behavior of the overall system. For this reason, it is important to understand the specific resource requirements of applied sensing technologies, and to actively prevent overstraining the available resources, e.g. by deactivating sensors that are currently not required, or do only provide marginal information. Although partial deactivation of sensors might possibly lead to a decrease in performance of certain parts of the system, it might be inevitable to maintain the global operability of the overall system.

Besides limitations that result from technical or physical resource constraints, additional limitations might be motivated by economical considerations. The designer of a system might want for instance to limit power consumption to extend battery life time of mobile devices, or limit CPU power to allow for the use of cheaper and/or lower-voltage microprocessors. Although motivated differently, economical resource constraints have the same effects than physical constrains; they limit the set of sensors that can be concurrently operated at any given point in time.

In the following subsections we give examples of resource-constrained sensing technologies that are typically used in instrumented environments. The basic properties of each technology are introduced, and the most relevant resource constrains are discussed. We then introduce the general problem of sensor selection given limited resources, which is to identify the "most promising" subset of sensors which can be simultaneously operated without exceeding a given resources limit. The sensor selection problem provides the main motivation for the utility model for observation information in plan recognition applications, that we present in chapter 7.

2.3.1 Wireless Sensor Networks

Wireless sensor networks (WSNs) consist of spatially distributed autonomous devices, so-called *sensor network nodes* (also called *motes*). Each node is

equipped with a power supply (usually a battery), a microprocessor, memory, and a radio. Some or all nodes of the network are equipped with sensors to *collaboratively monitor environmental conditions*, such as temperature, light, humidity, sound, acceleration, pressure, or air pollution. Via their radios nodes can communicate, and *multi-hop routing protocols* [SLL+08] allow to transmit information over large distances, e.g. to static *gateways* which interface a wired infrastructure (see Figure 2.2). The development of WSNs was originally motivated by military applications such as battlefield surveillance [BHK+06]. Today, WSNs are also used in many civilian application areas, including environment and habitat monitoring [MCP+02], healthcare applications [AVH09], home automation [HKB+07], and traffic control [CCCT05].

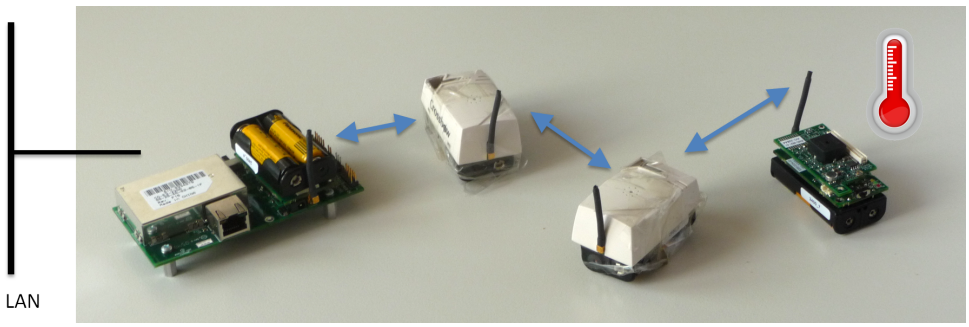
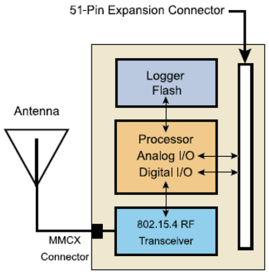


Figure 2.2: Multi-hop routing of temperature measures in a wireless sensor network towards a LAN gateway.

Existing wireless sensor networks come in multiple different designs and make use of a broad range of different technologies. A popular and widespread WSN system is Crossbow's *MICAz* [Cro09b]. Figure 2.3 shows a block diagram of the base nodes function units and summarizes the technical details of the MICAz system. MICAz motes have been used to implement the instrumented whisk and control cube interface in our *Smart Kitchen* environment (see chapter 9). Other examples of existing WSN systems are IMotes [Cro09a], Sun SPOTs [SHDC06], and μParts [BKR+06].

Resource Constraints

Depending on the concrete application scenario, the allowed size and cost of sensor network nodes are usually constrained. These constraints result in corresponding constraints on the availability of resources such as energy, memory, computational power, and bandwidth [RM04].



Processor/Radio Board	MPR2400CA	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	> 100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	Digital I/O,I2C,SPI	
RF Transceiver		
Frequency band[1]	2400 MHz to 2483.5 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 V - 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green and yellow
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Figure 2.3: Technical specification of MICAz motes (pictures and data taken from [Cro09b]).

One of the most critical resources in wireless sensor networks is energy, which obviously is a limited resource in battery-driven nodes. As the primary mean of operation it affects respectively is affected by the utilization of most other resources: Every operation – even a simple sensing or computation operation – drains valuable power from the batteries. Landsiedel and his colleagues have investigated energy consumption of MICA2 motes [LWG05], which are similar to the MICAz modes presented above. Table 2.1 exemplary shows the currents drawn by some of the considered operations.

Device	Condition	Current
CPU	Active	7.6 mA
	Idle	3.3 mA
	Standby	237.0 μA
LED	Active	2.2 mA
Sensor Board	Active	0.7 mA

Device	Condition	Current
Radio (900 MHz)	Core	60.0 μA
	Rx	9.6 mA
	Tx (+5 dBm)	26.9 mA
	Tx (0 dBm)	17.0 mA
	Tx (−18 dBm)	8.8 mA

Table 2.1: Currents draw by different operations performed on a MICA2 wireless sensor network node (data taken from [LWG05]).

The numbers in Table 2.1 illustrate that the overall power consumption and thus the resulting battery life time heavily depends on the concrete use of the sensor node: How often is the CPU idle, how often are the sensors activated, and how often is data received and send via the radio link. These factors can be controlled by the applications that utilize the wireless sensor network, hence applications to some extend can influence how fast the batteries get drained.

The degree to which energy is a finite resource depends on the concrete application scenario. If we assume, that a technician is available 24 hours a day to replace empty batteries, and ignore the fact that there is a minimum time which he needs to do so, one can understand energy as a nearly infinite (although very expensive) resource. In practical applications of course this assumption is usually not reasonable. Most often, one might want to define an application-dependent minimum life span for the batteries to last, e.g. for practical or economical reasons. The required life span then limits the electrical power that applications might consume in a given time period.

New technologies that have been introduced lately try to reduce the dependency of electrically operated mobile devices on batteries for power supply. *Power casting* for instances wirelessly transmits electric energy to mobile devices over short distances via inductive coupling. *Energy harvesting* is the process of deriving electrical energy from ambient energy that is naturally present in the environment like solar, thermal, kinetic, or electromagnetic energy. However, both technologies at their current state of development are not able to solve the problem of limited energy supply in general mobile applications: Power casting only works over short distances of up to one meter, while energy harvesting currently is only able to derive small amounts of energy. Hence, energy resource management still is an important issue in such systems [Mos09].

A second limited resource in WSNs is communication bandwidth. As all nodes usually communicate via the same frequency, spatially close nodes have to share the available carrier bandwidth. A common protocol for communication in WSNs is IEEE 802.15.4 (i.e. used by ZigBee [Zig08]), where bandwidth is limited to 250 kbit/s. The resource "communication bandwidth" in many respects differs from the resource "energy". First of all, energy usually is a *local resource*; if the batteries of one node are exhausted, other nodes still can function. Communication bandwidth is a *global resource*; if communication bandwidth is exhausted, all nodes in a particular area are prevented from communicating. Secondly, while it is possible to save energy for later use, we cannot save communication bandwidth. In this sense, communication bandwidth is a finite, but otherwise free resource (although communication itself might consume other costly resources like energy).

Energy and communication bandwidth are only two examples for constrained resources in WSNs. Further examples include CPU power, working memory, or resources consumed by specific sensors attached to nodes.

2.3.2 Radio-Frequency Identification

Radio-frequency identification (RFID) allows for the contactless *identification and tracking* of physical objects via radio fields. A reader device in the environment queries small transponders (also called *tags*) attached to objects and/or locations, which return information like a *unique ID number*. In contrast to marker-based systems that rely on optical sensing like bar codes, RFID does not require an unobstructed line of sight between reader and tag. This allows to unobtrusively *integrate RFID tags and readers into objects*, which makes identification more robust towards mechanical damage. Further advantages over optical systems are that tags do not have to be exactly align to the reader, and the possibility of *reading multiple tags at once*. An early form of RFID was originally developed for military "friend or foe" identification. Today, RFID is mostly used to track items in logistic processes [SKSM07], for the purpose of indoor positioning [BS05], or to track objects and interactions in novel kinds of user interfaces for smart home [Sch07a] or edutainment [NGK+05] environments.

Figure 2.4 shows the main components that comprise an RFID system: The *RFID reader* generates an RF signal, which is emitted by an *RFID antenna*. *RFID tags* which are in range of this signal receive and process the request, and send back the response to the reader. Besides a read-only *globally unique identification number*, current tags often carry a limited amount of *persistent memory*, which can be remotely read and reprogrammed by the reader. A new trend it to equip tags with *sensors and enhanced processing capabilities* like encryption, thus turning them into so-called *smart labels* or *smart items*.

RFID systems are primarily distinguished based on the kind of energy supply that is used to power the transponders. *Passive RFID* tags utilize the energy of the radio field created by the reader for processing and communicating, and thus do not require an own energy source. The lack of a battery allows for low production costs and nearly unlimited life time. The main drawbacks of passive transponders are a rather low reading range and the tags' inability to function outside of a reader's antenna field. *Active RFID* tags are equipped with an autonomous power supply (usually a battery). They allow for larger reading ranges and can continuously operate even in the absence of a reader. The latter point is relevant in data logging applications, where tags are equipped with additional sensors to actively monitor

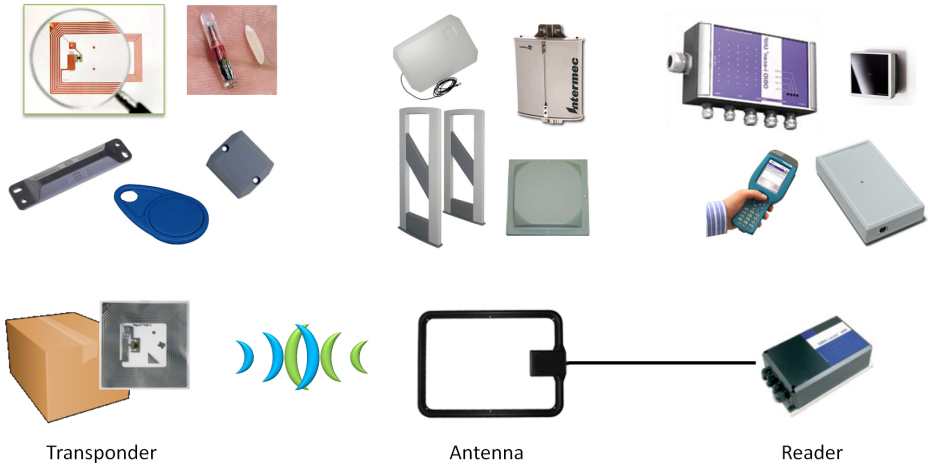


Figure 2.4: Components of an RFID system: Transponder (left) and reader (right) communicate with each other via an RF field emitted by an antenna (middle). Components might come in different sizes and configurations (top).

environmental conditions like temperature or pressure. Drawbacks of active RFID tags are larger form factor, higher production cost, and limited life span due to drained batteries. *Semi-Active RFID* tags are equipped with a battery for continuous sensing, but use a passive approach for communicating. As such, they provide much longer battery life for sensing, while at the same time allowing for unlimited communication and memory access. Drawbacks are the high price and low reading range. The characteristics of passive, active, and semi-active RFID systems are summarized in Table 2.2.

	Passive RFID	**Active RFID**	**Semi-Active RFID** [1] for sensors [2] for radio
Tag Power Supply	reader field	internal battery	battery[1], reader field[2]
Range of Operation	in reader field	everywhere	everywhere
Reading Range	≤ 7 m	≤ 100 m	≤ 7 m
Cost per Tag	~ 0.50 EUR	~ 20 EUR	~ 20 EUR
Life Time	unlimited	~ 2 years	limited[1], unlimited[2]

Table 2.2: Classification of RFID systems based on the tags' energy source.

In the *Smart Kitchen* environment presented in chapter 9 passive RFID is applied to detect the ingredients and tools that are used during cooking.

Resource Constraints

The primary limited resource in RFID applications is the radio communication channel. As all readers of the same type share a common frequency for communication, close-by readers easily interfere each other and thus usually cannot be operated at the same time. The concrete degree of interference depends on the individual setup and the used technology, but the problem is relevant in most practical RFID-based applications.

An example that illustrates this problem is the experimentation table of the *VirtualConstructor* (COHIBIT) exhibit [NGK+05] (see Figure 2.5, left side). In this edutainment exhibit, visitors can assemble physical 3D models of imaginary cars from a set of given modules like engine compartments, passenger cabins, roofs, and rears. Two human-like animated characters comment the visitors' actions and provide interesting information about the assembled car. In order to create the illusion of intelligent agents which "naturally" observe the user's actions, RFID tags are mounted unobtrusively inside the car modules. Hidden antennas in shelves and the central experimentation table allow the system to monitor the visitors' interactions with car modules and the resulting configuration of the constructed car. With a total of 15 RFID readers distributed over an area of only 0.5 square meters, the highest density of overlapping RFID fields is reached on the surface of the experimentation table (see Figure 2.5, right side).

The problem of overlapping radio fields in RFID applications can be solved if only one antenna at a time is activated in a round-robin fashion. This can either be achieved by attaching multiple antennas to a single reader via hardware multiplexers, or by turning off the RF signals of individual readers via software commands. On the downside, both approaches reduce the recognition rate and may result in delayed recognition of slow-moving objects, or even the total miss of fast-moving objects. An alternative solution (if supported by the used readers) is to adjust output RF power, which reduces recognition range and potential interferences with other readers. This effect can be used to choose a power level which provides the best trade-off between reading range and interferences.

Another relevant resource of RFID systems is electric energy. In active RFID systems, every reading cycle drains power from a transponder's battery. One might therefore want to limit the number of reading cycles to extend the battery life time of active transponders. For active and passive systems power consumption of RFID readers is an issue if they are used in mobile devices like RFID-enabled shopping carts (see subsection 5.1.3). Here a reduction of reading cycles also improves the system's operation time.

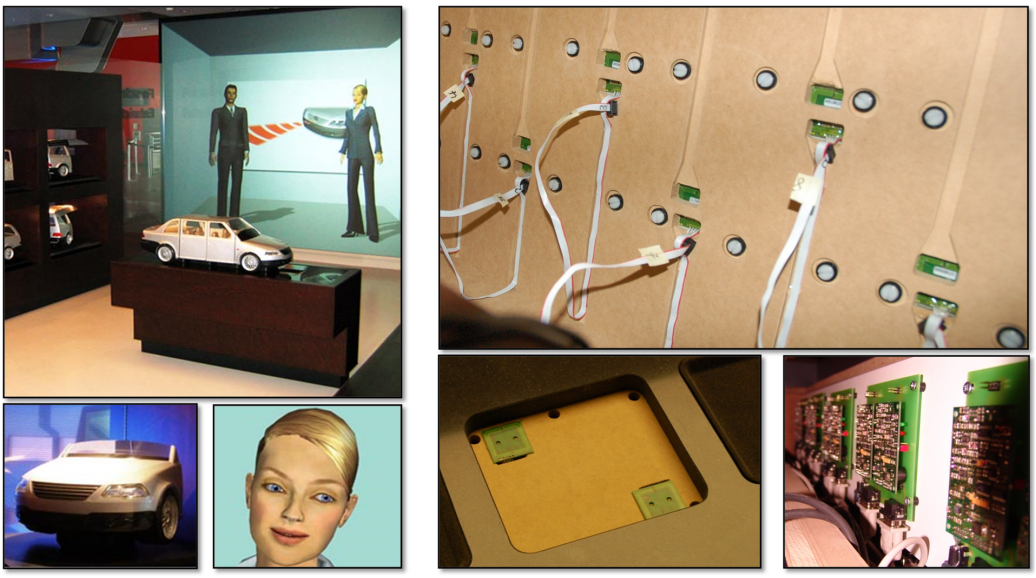


Figure 2.5: Resource conflicts arise in the COHIBIT exhibit (left) due to overlapping fields of near-by antennas in the experimentation table (right).

2.3.3 Computer Vision

Computer Vision comprises technologies that obtain information from image data. In instrumented environments computer vision can be used for a broad range of applications. An example is the assessment of the topology of an environment, including the *location of displays* [SBK05b] or objects [BSS04]. Other use cases are related to the detection of user actions like *selection and data manipulation gestures* over projected displays [SB05] or the observation of the user's *interactions with an instrumented shelf* [Kah07]. Further use cases include *face recognition* [BCF05] or the recognition of the user's center of attention through *gaze tracking* [MM05].

Figure 2.6 shows the general components of a vision-based sensing system. A *digital camera* records images or a video stream of the environment. Image and video format usually can be adapted (frame rate, resolution, color depth, compression ratio, etc.). Images are then transferred from the camera to a computer via USB, FireWire, Ethernet, or WiFi connections. On the computer, the image stream is analyzed by one or more *computer vision algorithms*. More sophisticated cameras allow to dynamically adjust some or all of their operation parameters, including the output image format or parameters which change the cameras physical field of view like pan, tilt, or focal length (zoom).

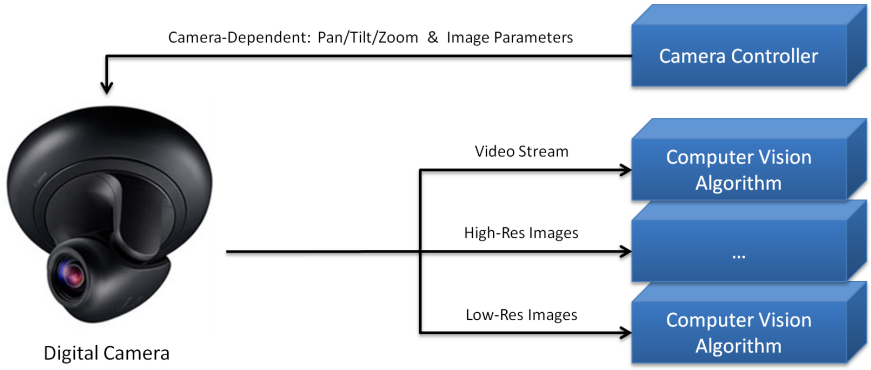


Figure 2.6: Components of a vision-based sensing system: A camera provides image streams, which are analyzed by computer vision algorithms. Camera parameters can be dynamically adjusted by a camera controller.

Recent trends in the area of computer vision involve systems comprised of multiple cameras (like stereo or multi-array cameras), very-high-resolution gigapixel cameras, and sphere cameras which allow capturing 360° images of an environment. All these approaches require the overall system to perform sophisticated (hence complex and expensive) processing of the camera data in order to derive meaningful information from the recorded images.

In our *Smart Kitchen* environment (see chapter 9) a set of digital video cameras is used to record a multi-angle video stream of the user's cooking process. Computer vision is used for selecting the camera image which currently shows the main action via motion recognition.

Resource Constraints

Computer vision often is resource intensive, as it requires the capturing, transmission, and processing of possibly huge amounts of data. This especially holds if high resolution, true color, and high frame rate image streams are considered. Stressed resources include communication bandwidth, working memory, and CPU time of involved components. Another limited resource is the field of view of steerable cameras. While such cameras can be orientated in nearly arbitrary direction, they can only observe a limited part of the environment at any given point in time.

A system which illustrates the effects of resource constraints in vision-based systems is *SearchLight* [BSS04], which realizes a search function for physical objects in instrumented environments. Figure 2.7 illustrates the general idea: A room (top left) is equipped with a steerable high-resolution

digital camera and projector unit on the ceiling (top center). Movable objects are tagged with optical markers (top right). The digital camera then takes overlapping pictures of the environment (bottom left) and applies computer vision based on the AR Toolkit library [KB04] to search for optical markers in these images. If a particular object is searched, the system retrieves from the list of found markers the correct orientation of the camera/projector unit and the markers position in the image, and uses this information to project a light cone at the position of the searched object (bottom right).

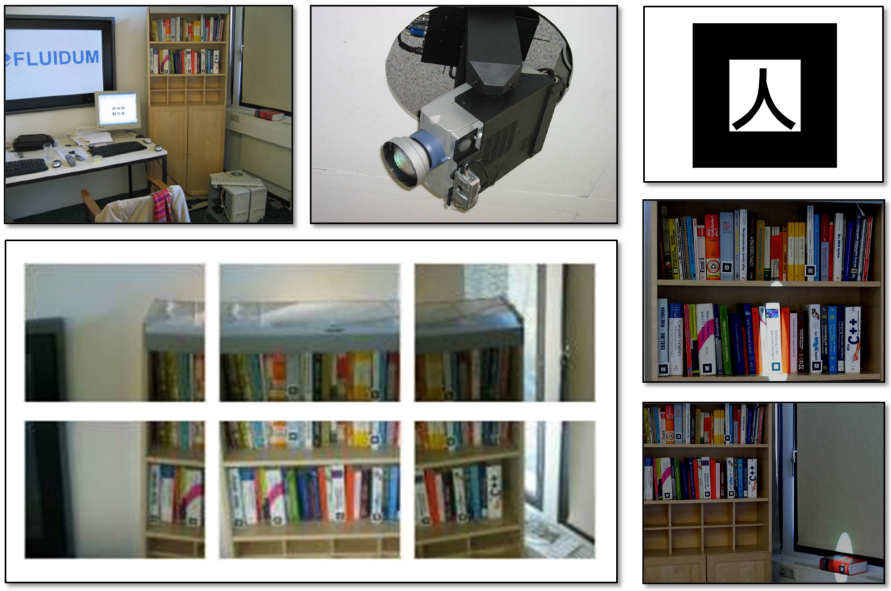


Figure 2.7: The *SearchLight* system realizes a search function for physical objects in instrumented environments. A camera/projector unit scans a room for objects tagged with optical markers (top row and bottom left) and then highlights searched objects (bottom right).

In *SearchLight*, different resource constraints affect the performance of the overall system. A critical factor is the amount of time that is required to scan the room. This time directly depends on the resolution of the taken images, which can be configured via the camera's API. On the one hand, changes to the resolution affect the resulting time for image transmission (the prototype camera has a slow USB 1.1 connection), as well as the runtime of the marker detection algorithm. On the other hand, the used resolution determines the minimum size of the markers, as higher images resolutions allow for the recognition of smaller markers. In order to optimize the scanning process, the image resolution has to be chosen such that it provides a good trade-off

between recognition performance given the actual size of used markers, and the required time to complete the scanning process.

Even more interesting with respect to resource constraints is the case of mixed scanning and projection operation. Here, the limited resource is the projector/camera unit's orientation, as it only can be oriented in one direction at a time. Here, resource conflicts arise if the projector should highlight an object in a particular area of the room, while at the same time the system is interested in what happened to a region in another area of the room, e.g. because the system got some information from sensors in a shelf that some object was moved (but without knowing the exact location). In this case, the system has to decide whether it is more important to interrupt projection in order to sense the shelf, or if it is sufficient to look at the shelf once projecting has finished. A similar problem is to decide in which order and how often to scan certain areas of the environment. Here, it might be reasonable to start with more dynamic areas, and scan them in shorter intervals than areas which change less often.

2.3.4 User-Based and Social Sensing

In the previous sections we discussed resource-constrained sensing in the context of technical sensor systems. If we understand the concept "sensor" in a more abstract sense as a *general source of information*, then also the *human user* can be understood as a special kind of sensor. If for instance no humidity sensor is available to detect rainfall, the system could simply ask the user via some sort of user interface whether it is raining or not. More importantly, a human can provide information that is difficult to assess with physical sensors, for instance regarding her own mood or mental state. As *User-Based Sensing* we understand the process of information gathering that directly involves a human user. User-based sensing occurs in different forms (see Figure 2.8).

Active user-based sensing actively queries the user for required information. In general, active user-based sensing is a three-stage process: (1) Drawing the user's attention to the system via some visual or acoustic prompt. (2) Presenting the request and receiving the response via some multimodal interface (cf. symmetric multimodality in [Wah03b, Wah03a]). (3) Allowing the user to confirm or correct her input, e.g. via confirmation buttons in a GUI, or by reading back input in a voice-based interface.

Passive user-based sensing waits for the user to provide the system with information. This kind of interaction is often used in cases where the user can provide optional information to adapt a system or to enhance the quality of a system. The general interaction pattern is similar to the case of active

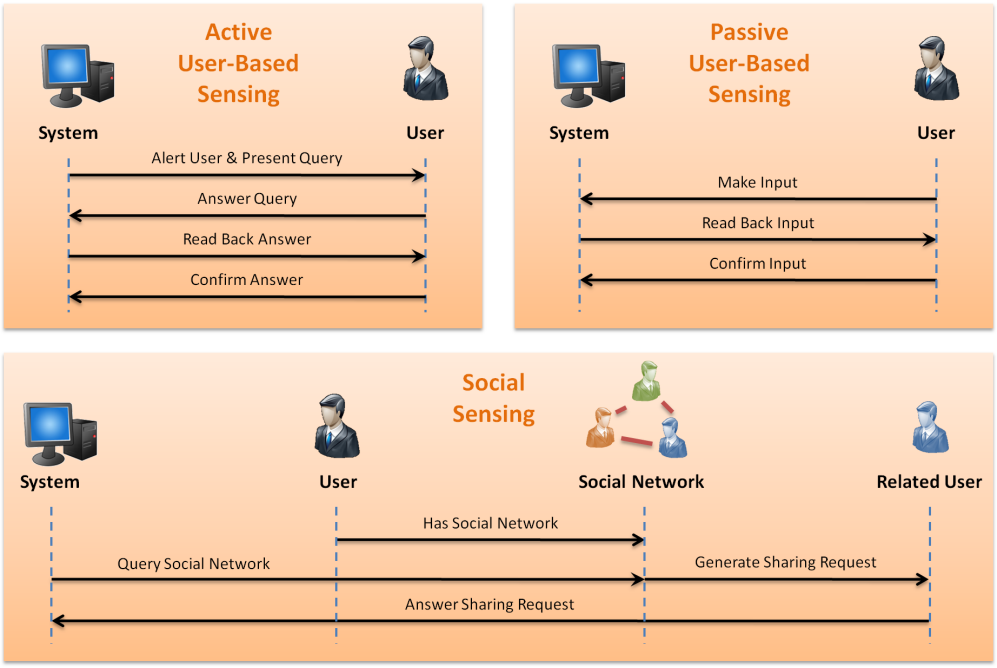


Figure 2.8: *User-based sensing* considers the user as a special kind of sensor that can be queried via multimodal user interfaces or social content sharing services. Three different kinds of user-based sensing exist.

user-based sensing, except that the initial interaction is initiated by the user through the invocation of some user interface function.

A third kind of user-based sensing is so-called *social sensing*. Social sensing does not query information directly from the user of the system, but from other users which are socially related to that user. An example use case of social sensing is a shopping assistant which evaluates information provided by other owners of the considered product, e.g. in form of ratings or reviews. A system that realizes such a social content sharing service is our SharedLife system [KSM09]. SharedLife automatically collects digital information about a user's experiences in everyday activities such as shopping [WKSB08] and cooking [Sch07a], stores this information as a set of semantic knowledge models that comprise the user's "augmented memory", and allows the user to share these memories with other users and applications. Users might select sharing partners from a buddy list or exploit social network data to find relevant and trustworthy sharing partners [KBSM07].

A practical example of social sensing are automated sharing requests for allergy information in the *Smart Kitchen* environment (see chapter 9).

Resource Constraints

Similar to technical sensors, the use of user-based sensing faces several resource constraints and is associated with costs for individual sensing actions. Here, the user's limited cognitive resources as well as existing social conventions are the main limiting factors. These include the user's attention, working memory, and the principle of "give and take".

Different kinds of user-based sensing consume different kinds and amounts of cognitive resources: In the case of active sensing, the user's attention needs to be drawn to the query prompt. As attention is a limited resource, this might require the user to withdraw attention from other tasks she is possibly following. Another important cognitive resource in user-based sensing is limited human working memory. In a prompt the user is often provided with several options to choose from. Results from psychology suggest, that the human working memory on average can only keep around *seven option alternatives* at the same time [Mil56]. Thus, users should not be provided with more than seven alternatives at a time to allow for the efficient processing of the query. If more than seven options exist, the system should decide on the seven most important ones to query first, and present other options in deeper menus or at a later point in time.

A resource which is less of cognitive nature but very well limited is the user's patience. If the user is repeatedly interrupted and asked to answer questions which seem unnecessary to her, she might sooner or later get annoyed by the system and finally might refuse to cooperate. In order to prevent cognitive resources like attention or working memory and non-cognitive resources like the user's patience from being exhausted by user-based sensing, the system should distinguish between important and unimportant queries given the current context and available resources. Thus, one should limit the number of user interruptions and distraction to the necessary minimum.

In the case of social sensing, additional constraints might be introduced by existing social conventions. An important convention for instance is not to annoy potential sharing partners by flooding the social network with requests. A related rule is to observe the law of "give and take": A sharing partner who from time to time receives information from us might be more willing to answer our requests than a user, which we asked several times before but with whom we never shared any of our own content. Moreover, users might feel "interrogated" and eventually may block access if they are faced with large amounts of undirected queries.

For the reasons given above, users and systems should carefully select the most promising sharing partners and ask these first, instead of querying random users by chance in the search for a particular information item. In

addition, for each query the potential costs and benefits have to be traded off. If the user wants to surprise some person with a birthday cake, it might be a bad idea to query this user's memory for her favorite cake one day before.

2.4 Sensor Selection

Whenever operating all sensors in an environment at the same time is not possible or desired due to physical or economical resource constraints, the system has to decide which sensors to activate. The challenge then is to find the subset of sensors which promises to deliver the most "valuable" or useful information, and whose total resource consumption at the same time does not exceed the given available maximum of resources. This problem is known as the general problem of *sensor selection* under limited resources.

By performing sensor selection, systems can adapt their operation to the amount of available resources. Wahlster and Tack distinguish three classes of system that perform resource adaptation [WT97]: *Resource-adapted* systems are optimized to deal with a fixed and known set of limited resources. Here, the output quality only depends on the input quality. *Resource-adaptive* systems use a fixed strategy for dealing with unknown resource constraints, and *resource-adapting* systems vary their strategy based on the available resources. The output quality in the two latter cases depends on the input quality and the amount of available resources. In this sense, systems that perform sensor selection are resource-adapting systems, as they dynamically chose the best set of sensors based on the amount of available resources.

2.4.1 Utility-Based Sensor Selection

Different ways exist to approach the problem of sensor selection. A discussion of related work in the area of sensor selection is provided in section 5.3. In the following we assume an application-independent definition which requires, that an application-dependent *utility function* is given, which allows us to judge the expected utility of different subsets of sensors with respect to the considered higher-order application (in the context of this thesis, this application is plan recognition). The primary objective of utility-based sensor selection then is to find a subset of sensors which maximizes the value of this function while respecting the given resource constraints.

We now formally define the problem of utility-based sensor selection. The presented definition assumes that the resource consumption of individual sensors is described by a known *resource consumption function*, and that resource consumption is additive, which means that the total resource con-

sumption of a set of sensors equals the sum of the individual sensors' resource consumptions[1]. For a set of n constrained resources, amounts of available and consumed resources are expressed as vectors from $\mathbb{R}^n$, where each element of the vector corresponds to a particular resource.

Definition 2.4.1. A **sensor selection problem** is a tuple $(\mathcal{S}, R, u, r, \hat{r})$, where

- $\mathcal{S}$ is a finite *set of sensors*.
- $R = \{R_1, \ldots, R_n\}$ is a finite *set of resources*.
- $u : 2^{\mathcal{S}} \mapsto \mathbb{R}$ is the *sensor utility function*, where $u(\mathcal{S}')$ denotes the expected utility of activating a subset $\mathcal{S}' \subseteq \mathcal{S}$ of sensors ($2^{\mathcal{S}}$ denotes the power set of $\mathcal{S}$).
- $r : \mathcal{S} \mapsto \mathbb{R}^n$ is the *resource consumption function*, where $r(s)$ returns a vector of size n, where the n-th element denotes the amount of resource R_n which is consumed while running sensor $s \in \mathcal{S}$.
- $\hat{r} \in \mathbb{R}^n$ is the *maximum available amount of resources*, where the n-th element of $\hat{r}$ denotes the maximum amount available of R_n.

The solution of a sensor selection problem is a set $\mathcal{S}^*$ of sensors, which maximizes the expected utility while not exceeding the given resource limit. In order to realize this utility in a practical setting, we then have to activate all sensors that are part of $\mathcal{S}^*$, and deactivating all other sensors (all sensors not in $\mathcal{S}^*$). Formally, sensor selection is the following optimization problem:

Definition 2.4.2. The **solution** of a sensor selection problem $(\mathcal{S}, R, u, r, \hat{r})$ is a subset $\mathcal{S}^* \subseteq S$ of sensors, where

$$\mathcal{S}^* = \arg\max_{\mathcal{S}' \subseteq \mathcal{S}} u(\mathcal{S}') \text{ with } \sum_{s \in \mathcal{S}'} r(s) \leq \hat{r}$$

[1] If resource consumption is not additive, e.g. because synergies between different sensors exist which lower the resulting total resource consumption of a group of sensors, the resource consumption function has to be defined over sets of sensors.

2.4.2 Representing Configurable Sensors

The sensor selection problem formulated above only considers *binary sensor selection*, in which each sensor is either activated or completely deactivated. In practical applications, we often have to deal with *configurable sensors*. Configurable sensors can be operated in several differing *configurations*, where each configuration might consume different kinds and amounts of resources, and in turn delivers sensor data with varying utility. An example is an RFID reader/antenna unit whose antenna power level can be dynamically controlled at runtime. Depending on the selected power level, reading range and energy consumption change, as well as the interferences that result for other near-by readers (see subsection 2.3.2). Another example are steerable cameras, or cameras which allow to adjust the image format.

If configurable sensors exist in an environment, sensor selection should not only consider switching such sensors on or off, but also operating them in different configurations. Accordingly, the solution of the sensor selection problem should for every active sensor provide the best configuration (or set of configurations, if the sensors allows to activate more than one configuration at a time[2]) to use. In the context of configurable sensors, sensor selection is also called *active sensing*.

Formally we can describe configurable sensors as sets of configurations. Let s_i be a configurable sensor with n_s configurations $C_s = \{c_{i,1}, \ldots, c_{i,n_s}\}$. As different configurations of a sensor might consume different types and/or amounts of resources, we have to define the resource consumption function based on configurations. To distinguish it from the sensor-based function r in Definition 2.4.1, we denote the configuration-aware resource consumption function r_C. Similar holds for the utility function. As different configurations of a sensor might provide different utility of information, our utility function has to be defined based on configurations (and hence is denoted u_C).

The problem of *configurable sensor selection* for a set $\mathcal{S}$ of configurable sensors then is to choose from the set of all configurations $C = \cup_{s \in \mathcal{S}} C_s$ a subset $C^* \subseteq C$, such that

1. $\sum_{c \in C^*} r_C(c) \leq \hat{r}$
2. $u_C(C^*)$ is maximized
3. for every sensor $s \in \mathcal{S}$ at most the maximum allowed number of concurrent configurations is selected from C_s

[2]Most sensors only support single configurations at a time. E.g. a regular steerable camera can only look into exactly one direction. A counterexample is a radio receiver with a dual tuner which can be configured to listen on two frequencies simultaneously.

Except for the additional third condition, the problem of configurable sensor selection is very similar to the general problem of sensor selection as defined in the previous subsection (cf. Definitions 2.4.1 and 2.4.2). In order to solve the configurable sensor selection problem we can either extend existing sensor selection algorithms to account for the additional third constraint, or use the following trick to encode the additional constraint in the resource consumption function r and resource bound $\hat{r}$, and then apply standard sensor selection algorithms to solve the problem of configurable sensor selection.

The definition of sensor selection given above can be applied to the case of configurable sensors if we consider every configuration to be an individual "virtual" sensor. Hence the set of all configurations C becomes the set of all virtual sensors. Next, we extend the set R of resources by one additional "virtual" resource R_s for each configurable sensor s. The *extended upper resource bound* $\hat{r}'$ then is chosen in a way that the maximum amount of R_s represents the number of configurations of sensor s that can be activated at the same time. If $\hat{r} \in \mathbb{R}^m$ then the resulting extended resource bound $\hat{r}' \in \mathbb{R}^{m+n}$ is defined as

$$\hat{r}' = \hat{r} \circ [a_{s_1}, \ldots, a_{s_n}]$$

where $\circ$ is the concatenation operator which concatenates two vectors of size m and n to a single vector of size $m + n$, and a_s is the number of configurations that can be active for sensor s at the same time.

Next we chose the *extended resource consumption function* in a way that for each configuration $c_{i,j}$ belonging to configurable sensor s_i one unit of R_{s_i} is consumed. Formally, $r'_C : C \mapsto \mathbb{R}^{m+n}$ then is defined as

$$r'_C(c_{i,j}) = r_C(c_{i,j}) \circ [0^{i-1}] \circ [1] \circ [0^{n-i}]$$

where $[0^n]$ denotes the vector from $\mathbb{R}^n$ where all elements equal 0. With the above considerations, the problem of configurable sensor selection can be formulated as the following general sensor selection problem according to Definition 2.4.1:

$$(C, R \cup \{R_{s_1}, \ldots, R_{s_n}\}, u_C, r'_C, \hat{r}')$$

With this formulation we can now use existing general sensor selection algorithms to solve the problem of configurable sensor selection respectively active sensing.

2.5 Summary

In this chapter we introduced the main application domain for the work presented in this thesis. We explained the need for sensor selection in resource-constrained instrumented environments and defined the general problem of utility-based sensor selection. This problem provides the main motivation for the development of the utility model for sensor information that we present in chapter 7. At the end of chapter 7 we present two sensor selection strategies that solve sensor selection problems of the form described by Definition 2.4.1. By exploiting the reformulation we presented in subsection 2.4.2, these algorithms can be used to additionally solve problems of configurable sensor selection in cases where sensors can be operated in more than one configuration.

If we knew what it was we were doing, it would not be called research, would it?
Albert Einstein (1879–1955)

3 Plan Recognition

This chapter introduces plan recognition, which provides the main focus of this thesis. We introduce and motivate the *general concept* of plan recognition (see section 3.1) and present a *formal definition* and *general algorithm* (see section 3.2). We present a classification scheme for plan recognition (see section 3.3) and define instrumented environments with plan recognition support (see section 3.5). We conclude with a discussions of *standard assumptions* made in plan recognition applications and a description of plan-related *terminology* that is used throughout the rest of this thesis (see section 3.6).

3.1 General Concept

Plan recognition is the process of inferring an agent's *goals and plans* by observing its behavior. In this context, the term "agent" can refer to living individuals like humans and animals, as well as to autonomous technical systems like robots and software agents. Typical plan recognition approaches take a sequence of observations as input and try to explain these observations through a set of higher-order plans. A plan is a *completely or partially ordered* list of actions which if executed achieve a particular desired goal. Knowledge about an agent's goals and plans is important in a broad range of applications, which can be roughly classified depending on their general purpose:

- **Assistive Applications** aim at supporting an agent in performing its tasks and reaching its goals through the provision of *proactive assistance*. Here, goal and plan information can be used to anticipate the agent's needs, to present situation-dependent information, to adapt the application or environment to the agent's task, or to execute operations on behalf of the observed agent. Assistive applications are typically situated in *work* and *home* environments [BBG06]. A related application area which has recently evolved is the support of elderly and mentally impaired users in *ambient assisted living* scenarios [RBBG07].

- **Defensive Applications** aim at hindering a hostile or competing agent in performing its tasks and reaching its goals through the *proactive taking of counter measures*. This ranges from triggering an alarm to the execution of actions which directly fend the opponent agent's actions. Defensive applications are often found in the *military domain*, e.g. to explain the behavior of enemy troops [HGP99]. Other defensive applications include public security applications that protect critical infrastructures, recognize and predict the actions of terrorists, or concern *intrusion detection* in computer networks [GG01]. A rather new application area is plan recognition in strategic computer games that allows opponent non-player agents to more intelligently react to the human player's plans and actions.

- **Scientific Applications** aim at understanding an agent's actions and goals for scientific reasons. Here, goal and plan knowledge might be used to identify and learn about relations between executed actions and higher-order goals in behavioral analysis, but do not directly aim at supporting or hindering the observed agent.

Plan recognition can be applied whenever information about an agent's goals and plans is required but unknown, and the involved agent cannot or should not be directly asked. The latter restriction obviously holds in defensive applications or the observation of non-human agents, but is also relevant in the context of cooperative human users: A human user might be unable to communicate its goals and plans to the system, either because this would require her to learn a formal language or because the cognitive load of interacting with the system or formulating the personal plans would be too high. In addition, it might not be desirable to directly ask the user, e.g. because this would alter her behavior or would distract the user from her actions. By passively observing an agent, plan recognition instead can unobtrusively gather information about goals and plans.

The process of plan recognition can be best explained with a simple example, in which we assume that plan recognition is performed by a human called Bob. Bob watches his wife Alice preparing dinner in the kitchen. He is curious about which dish Alice intends to prepare, so he carefully observes her actions:

> At the beginning of observing Alice's cooking, Bob has no evidence on the recipe that Alice is going to prepare. He might know from experience that certain dishes are generally preferred by Alice over others (and thus are more likely to be prepared), but he cannot generally restrict the set of possible plans.

> Now Alice starts with the preparation: In the first step, Bob observes that Alice is boiling spaghetti in a pot of hot water. He concludes that Alice is most likely preparing a spaghetti dish, although he is not sure up to now which one.
>
> In the second step, Bob observes that Alice is using tomatoes to prepare a tomato sauce. Alice only knows two spaghetti recipes with tomato sauce – Spaghetti napoli and spaghetti bolognese – so Bob concludes that Alice is preparing one of these two.
>
> Next, Bob observes that Alice is adding minced meat to the tomato sauce. Now it seems obvious to him, that Alice is preparing spaghetti bolognese.

Table 3.1 summarizes Bob's observations (second column) and the hypothesis about Alice's plan he inferred from this observations (third column). Whenever new evidence becomes available, Bob can narrow down his hypotheses. Technically, Bob performs plan recognition, as he infers hypotheses about Alice's plans and intentions by observing her actions.

Step	**Observation**	**Plan Hypothesis**
initial	-	all known recipes
1	Alice boils spaghetti	spaghetti dish
2	Alice takes tomatoes	spaghetti napoli or spaghetti bolognese
3	Alice adds minced meat	spaghetti bolognese

Table 3.1: A simple plan recognition example: Observations of an agent's actions in a kitchen and possible hypotheses about the intended dish.

3.2 Formal Definition and General Algorithm

The example given in the previous section already reveals some general properties of plan recognition systems: In order to be able to infer Alice's intentions, Bob must know about all recipes that Alice probably might prepare. The knowledge model that describes the set of all possible plans is usually called *plan library*. A trivial plan library might simply contain a *complete enumeration* of all plans. Often this approach is not feasible, either because of the sheer amount of possible plans or because the plan library is of infinite size, e.g. because it contains plans with loops. Instead of a simple enumeration, plan libraries hence usually have a *factorized representation*, from which

plan candidates are constructed as required. For this one can use *generative systems* like grammars or general rule-based systems. Besides the plan library, a plan recognition system might need additional knowledge about the considered domain. In the previous example, Bob for instance has to know how a boiling pot of water looks like. The sum of additional knowledge that is required to perform plan recognition is called *background knowledge*.

Plan recognition might come up with more than a single possible explanation for a sequence of observations. Hence, *plan hypotheses* usually comprise a whole set of *candidate plans*, which are often annotated with a rating on the likelihood of each candidate. These likelihoods account for the lack of complete knowledge about the agent's intentions during the plan recognition process. As more observations are made and considered by the plan recognition system, the hypothesis is iteratively refined until it eventually comprises only the actually executed plan (or set of plans).

The components of a plan recognition system, the refinement of hypotheses, and the resulting flow of information are illustrated in Figure 3.1. Formally we define a plan recognition system as:

Definition 3.2.1. A **plan recognition system** is a tuple $(L, K, O, \mathcal{H}_L, h_0, pr)$, where

- L is a *plan library*
- K is the required background knowledge
- O is the set of possible *observations*
- $\mathcal{H}_L$ is the set of possible *plan hypotheses* over library L
- $h_0 \in \mathcal{H}$ is the *initial plan hypothesis*
- $pr : \mathcal{H}_L \times \mathcal{O} \mapsto \mathcal{H}_L$ is the *plan recognition function*, which updates a given plan hypothesis based on new observation information

The general definition of plan recognition does not restrict the kind, structure, and representation of plan libraries, background knowledge, observations, and resulting plan hypotheses. These factors mainly depend on the applied theoretical framework, on the concrete application domain, as well as on the concrete implementation of the used plan recognition approach. The

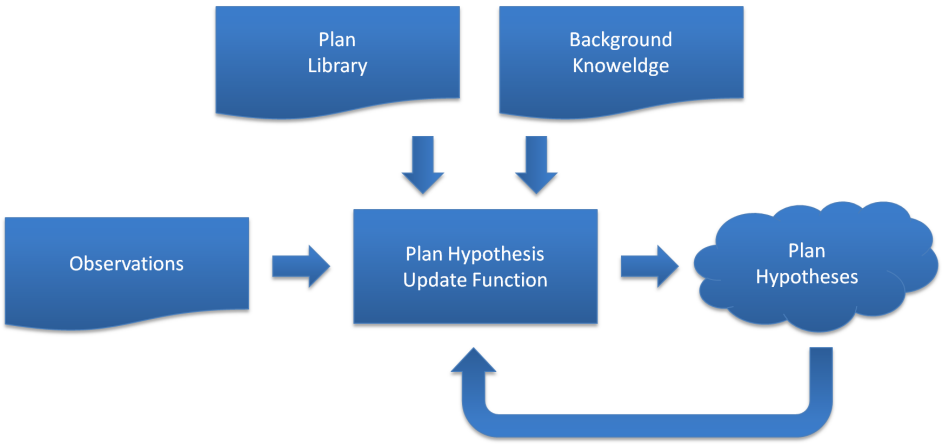


Figure 3.1: General Structure of a plan recognition system: Observation information is used to update the plan hypothesis. The update process usually requires background knowledge about the domain and the library of plans.

plan recognition approach that is developed in this thesis for instance uses a probabilistic automaton representation for the plan library, while background knowledge about sensors and resulting readings is formulated as a Bayesian network (see section 4.3). A detailed formal description of the plan recognition approach that is developed in this thesis is presented in chapter 6.

Given a plan recognition system $(L, K, O, \mathcal{H}_L, h_0, pr)$, the general process of plan recognition can be described by the following algorithm, where $h \in \mathcal{H}_L$ is the current plan hypothesis, $h' \in \mathcal{H}_L$ is the updated plan hypothesis, and $o \subseteq O$ is the set of current observations:

Algorithm 1: General Plan Recognition Algorithm

$h = h_0$
while *active* **do**
  $o =$ Collect observations
  $h' = pr(h, o)$
  Emit h'
  $h = h'$
end

Given a sequence $[o_1, \ldots, o_n]$ of observations, the general plan recognition algorithm generates a sequence $[h_1, \ldots, h_n]$ of plan hypotheses of same length, where $h_i = pr(h_{i-1}, o_i)$ for $1 \leq i \leq n$ (h_0 is provided as part of Defini-

tion 3.2.1). Hypothesis h_n then accounts for all information that was observed so far, and thus is the most specific hypothesis of all generated hypotheses.

The formulation given above assumes that the plan hypothesis update function *pr* is stateless, which implies that all eventually relevant information regarding past observations has to be entirely encoded in the current plan hypothesis h. Chapter 6 explains how this requirement is realized in the state-aware plan recognition approach that is proposed in this thesis.

3.3 Classification of Plan Recognition

In order to successful apply plan recognition techniques it is important to understand the agent's awareness of being observed, and its attitude towards the performed plan recognition process respectively the resulting use of plan recognition information. Both aspects generally depend on the concrete application scenario and directly influence the hardness of the resulting plan recognition problem. According to Waern [Wae96], three general subclasses of plan recognition problems can be distinguished in practical applications:

- **Keyhole Plan Recognition** describes the case where the agent is either unaware of the plan recognition process or aware but indifferent towards being observed. In this case, it neither actively supports nor hinders plan recognition, but behaves exactly as it would do without being observed. Keyhole plan recognition is not trivial to perform from the plan recognition system's perspective, as it requires the system to deal with possibly contradicting and/or incomplete information. For the observed agent the application of keyhole plan recognition does not introduce any extra efforts and thus supports the idea of instrumented environments to provide transparent proactive assistance. Keyhole plan recognition is mostly considered in *assistive* and *scientific applications*.

- **Intended Plan Recognition** describes the case where the agent is aware of the plan recognition process and has a positive attitude towards it. Hence he might intentionally choose his actions in a way that makes it easy for the plan recognition system to conclude his plans and intentions. From the system's perspective, intended plan recognition is easier to perform than keyhole plan recognition, as it tries to avoid ambiguous observations. From the observed agent's perspective, intended plan recognition requires more efforts than keyhole plan recognition, as it requires the observed agent to gain some understanding of the way the applied plan recognition system works, and then to choose its actions accordingly. For this reason, it partially contradicts the idea of

transparent, proactive assistance and is more similar to the traditional approach of direct and explicit interaction. Intended plan recognition is generally only considered in very specific *assistive applications*; mostly in cases, in which a low rate of recognition errors is required.

- **Obscured Plan Recognition** describes the case where the agent is aware of the plan recognition process, but has a negative attitude towards it. In this case, the agent might intentionally choose his actions in a way that makes it hard for the plan recognition system to conclude his intentions. Obscured plan recognition is even more hard for the system to perform than keyhole plan recognition. At the same time, obscured plan recognition also requires extra efforts on the side of the observed agent. Obscured plan recognition is mostly assumed in *defensive applications*, in which hostile agents do not want their plans to become apparent.

The most important properties of the three classes of plan recognition problems are summarized in Table 3.2. The vast majority of plan recognition approaches that exist today (see section 5.2) focus on the case of *keyhole plan recognition*. Keyhole plan recognition is also considered in the plan recognition approach that is presented in chapter 6 of this thesis. Keyhole plan recognition was chosen, as the keyhole assumption is valid in most of today's practical application scenarios. A concrete example of an assistive application of keyhole plan recognition is the *Semantic Cookbook* application hosted in an instrumented kitchen environment, which is presented in chapter 9.

Agent is Aware of Observation	**Agent's Attitude Towards Observation**	**Classification of Plan Recognition Problem**
no	n/a	keyhole plan recognition
yes	indifferent	keyhole plan recognition
yes	positive/supportive	intended plan recognition
yes	negative/obstructive	obscured plan recognition

Table 3.2: Classification of plan recognition problems according to the observed agent's awareness of and attitude towards the plan recognition process.

It should be noted, that although the plan recognition approach presented in chapter 6 assumes keyhole plan recognition, the utility model for observation information that is presented in chapter 7 is independent of the kind of plan recognition that is performed, and thus can also be applied in the cases of intended or obscured plan recognition.

3.4 Recognition of Overlapping Plans

Depending on the application domain and its current context an agent might follow more than one plan at a time. This case is called *overlapping plan execution* and has three special cases (see Figure 3.2):

- **Parallel Plan Execution** describes the case where two or more plans are executed independently of each other (Figure 3.2.a). An example is cleaning vegetables while listening to the radio.
- **Shared Plan Execution** is a special case of parallel plan execution in which a single action at the same time contributes to two or more plans (Figure 3.2.b). An example is turning on the oven if the user in parallel prepares a cake and a casserole.
- **Interleaved Plan Execution** means the execution of two plans where the execution of one plan is suspended while the second plan is executed (Figure 3.2.c). An example is answering a phone call while cleaning vegetables if the phone is located outside the kitchen.

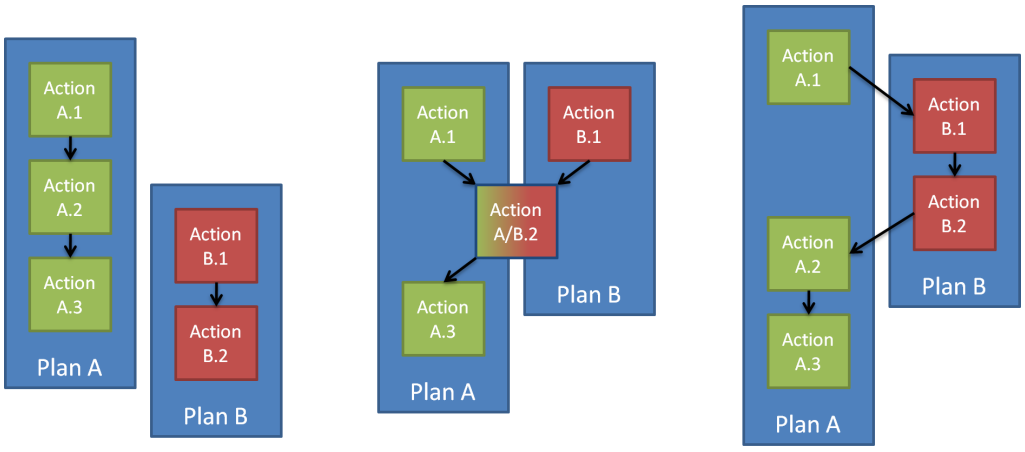


Figure 3.2: Parallel plans (a), shared plans (b), and interleaved plans (c).

The complexity of the plan structures that should be recognized by a plan recognition system determines the number of hypothesis that a complete plan recognition system has to consider. This at the same time determines the computational complexity of the concrete plan recognition problem (see subsection 5.2.7). Hence different plan recognition approaches might support different degrees of plan parallelism.

3.5 Plan Recognition in Instrumented Environments

The motivation of this thesis is the application of plan recognition in instrumented environments. The general use case of plan recognition in instrumented environments is shown in Figure 3.3. An agent is observed during the *execution of plans* in an environment E. The agent's behavior (respectively its effect on the state of the environment) is observed by a *set of sensors*, which is part of the environment's instrumentation I. Made observations are used by a *plan recognition* component to infer and update a *plan hypothesis*, which represents the system's belief about the currently executed plan. This hypothesis is used by a *support system* to proactively trigger the execution of value-added services. These services usually aim at supporting the agent in executing its tasks (other kinds of services are discussed in subsection 7.2.2). Plan recognition component, plan hypothesis, and support system are part of the environment's virtual layer D. The support is realized by a *set of actuators* that is part of instrumentation I.

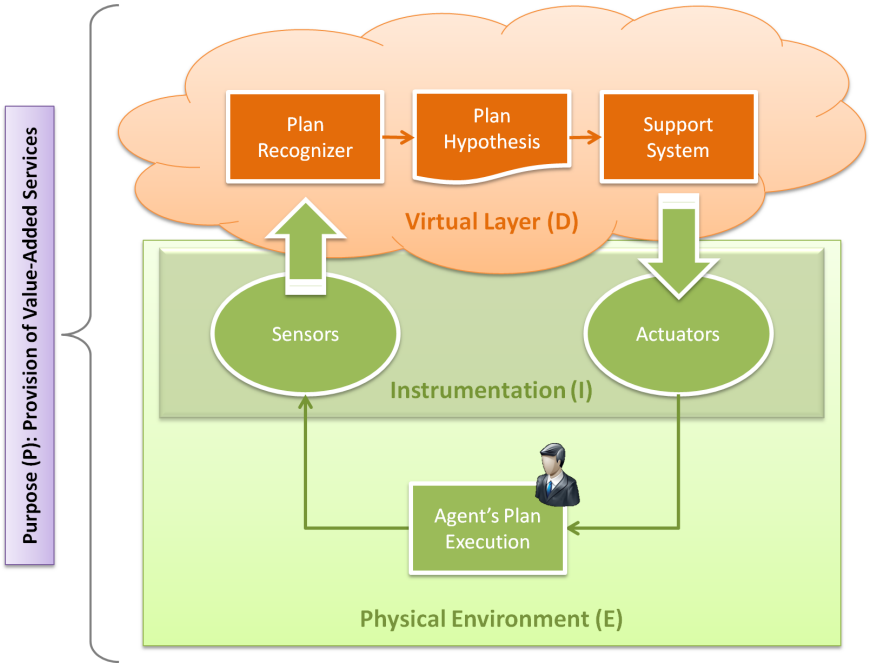


Figure 3.3: General view on the components that are involved in the realization of plan-recognition-based assistance in an instrumented environment (E, P, D, I).

The formal definition of *instrumented environments with plan recognition support* is based on the general definition of instrumented environments (see Definition 2.2.1) but adds a description of the components that are required to realize the general plan recognition and support functionality. Note that this definition assumes that plan recognition is performed to serve some higher-order goal (like user support), and in particular is not performed for the sake of its own. This assumption will become important in chapter 7, when we present our utility model for observation information in plan recognition applications.

Definition 3.5.1. An **instrumented environment with plan recognition support** for some set of user plans L is an instrumented environment (E, P, D, I), where

- P includes to proactively provide added-value services on the execution of plans from L.
- There exists a set $\mathcal{S} \subset I$ of sensors which (at least partially) allow to observe the execution of plans from L, and a set $\mathcal{B} \subset I$ of actuators which allow to realize added-value services as given in P.
- There exists a *plan hypothesis* $h \in D$, which represents the system's current knowledge about the agent's plan execution process.
- There exists a *plan recognition system* $(L', K, O, \mathcal{H}_L, h_0, pr) \in D$, which updates plan hypothesis h, where $K \in D$, $L' \subseteq L$, and $O \subseteq O_{\mathcal{S}}$ ($O_{\mathcal{S}}$ is the power set of all readings that might originate from sensors $\mathcal{S}$).
- There exists a *support system component* in D, which based on h realizes added-value services through actuators $\mathcal{B}$.

3.6 Standard Assumptions and Notation

Plan recognition systems usually make a set of standard assumptions in order to make the problem of plan recognition theoretically and computationally tractable. These most common assumptions are described in the following:

- **Availability of Meaningful Observations:** The most important and most obvious assumption is, that significant and meaningful observation information regarding the plans that should be observed is available in the environment. This usually means that suitable sensors are installed. Without such information, a plan recognition system has no chance to infer any reasonable hypotheses, which makes the application of plan recognition pointless.

- **Completeness of Plan Library:** The plan library provides the "blueprints" for the explanation of observations and the generation of hypotheses. As a consequence, the plan library has to represent all plans one might want to recognize. This includes all variants and even incorrect variants of correct plans, if the system should recognize these too, i.e. in order to provide assistance on how to fix an incorrectly executed plan. Some plan recognition systems even require the plan library to represent *all possible* plans, and not only plans that should be recognized. The plan recognition approach presented in this thesis assumes that all plans that should be recognized are represented by the plan library.

- **Purposeful Acting:** It is generally difficult for a plan recognition system to distinguish between actions that are executed purposefully (and thus have to be considered in the plan recognition system's reasoning), and actions that are executed without a specific purpose (and thus have to be ignored, as they are not part of the actually executed plan). Many plan recognition system therefore assume that all actions are purposeful executed. Section 8.4 shows how the plan recognition system presented in this thesis can deal with unpurposeful actions.

- **Limited Number of Concurrent Plans:** A plan recognition system is usually unable to decide whether a new observation constitutes the beginning of a new (parallel or interleaved) plan, or continues an already started plan. In order to keep the resulting number of candidate plans in a hypothesis reasonably low, most plan recognition systems assume a minimal number or fixed upper bound of simultaneously executed plans. The plan recognition approach presented in this thesis

considers the recognition of single plans only, which is sufficient in many practical real-world applications. These plans can be interleaved by other plans, but the interleaving plans themselves are not recognized by our approach.

Next we introduce and explain a couple of frequently used terms related to plan recognition. We start with terms related to states, goals, and acting agents, continue with the execution of basic actions, and finally cover plan structures composed of multiple actions:

State A state describes specific aspects of the physical or virtual world. The complete set of all partial states provides the so-called *world state*. A computer program normally has immediate access to the states of the virtual world. States of the physical world can only be accessed indirectly though adequate sensors. Some state information like the observed agent's state of mind might even be unobservable at all with existing technology.

Goal A goal is a defined, desired (partial) state of the world. In most cases, this state differs from the actual state of the world. If not, we say that the goal is *reached*.

Agent An agent is an entity which proactively and autonomously pursues a set of individual goals. To reach these goals, an agent executes a sequence of actions (cf. term "action" below). In most cases, the term "agent" refers to a human, but it can also refer to other living individuals like animals or even to technical systems like software agents or autonomous robots.

User In the context of plan recognition, the terms "agent" and "user" are often used synonymously. In the context of this work, we use the term "user" to express that an agent is assumed to be human.

Sensor A sensor is an entity, which if queried assesses a finite subset of the world state. Sensors might be technical devices, software probes, or living beings like humans. Sensors might not be perfect, which means that the resulting assessment might differ from the true state for a variety of reasons. Hence sensor readings are often assigned probability values that describe the system's belief in the returned data. Sensors can have side effects, such that taking a measurement influences the state of the measured system.

Observation An observation is a piece of information that results from querying a sensor. It usually describes a defined subset of the world state at the time of measuring. Observations might also result from querying multiple sensors, in which case one additionally can apply techniques for sensor fusion [CD93] to extract higher-order features from raw sensor observations.

Action An action is the modification of the world state through an agent. Actions might cause an observable change in the state of the environment, but not all observable changes are caused by actions (imagine changes to the environment caused by an earthquake). Agents usually execute actions on purpose to reach a certain (intermediate) goal, but might also execute actions unconsciously, by reflex, or by accident.

Plan A plan describes a concrete goal and a partially ordered list of actions. Besides partial ordering a plan might require additional constraints to hold, e.g. temporal constraints like minimum or maximum delays between actions. If executed in a valid order with all constraints satisfied, the given actions are assumed to reach the goal associated with the considered plan.

Plan Library A plan library describes the set of all known plans. Sometimes it is required to distinguish between the observed agent's plan library and the plan recognition system's library, which might or might not be the same.

Plan Hypothesis A plan hypothesis represents the plan recognition system's belief about the currently executed plan or set of plans. For this purpose plan hypotheses often make references to the assumed plan library.

3.7 Summary

In this section we introduced and formally defined plan recognition systems. The formal definitions provide the foundation for the definition of the state-aware plan recognition approach that we present in chapter 6. We further gave a formal definition of instrumented environments with plan recognition support, which provides the context for the definition of the utility model for sensor information in real-world plan recognition applications described in chapter 7. At the end of this chapter we discussed the assumptions we make in the following and introduced the notion that we use for the rest of this thesis.

I have hardly ever known a mathematician who was capable of reasoning.

Plato (428–348 BC)

4

Excursus: Probabilistic Reasoning and Decision Theory

In this chapter, we give a short excursus on probabilistic reasoning and decision-making under uncertainty. The theoretical frameworks presented in the following sections provide the foundation for the state-aware plan recognition approach that we present in chapter 6, and the utility model for observation information in plan recognition applications that we present in chapter 7.

Our excursus starts by motivating the need for reasoning under uncertainty (see section 4.1). Next we introduce the basic concepts and notations related to unconditional and conditional probabilities (see section 4.2). We continue with a presentation of Bayesian Networks as a graphical way of representing probabilistic knowledge (see section 4.3), and finish with a discussion of decision theory, which considers optimal decision making given incomplete knowledge (see section 4.4).

4.1 Motivation

Early computer systems which performed reasoning tasks used deductive systems like *first-order logic* to infer new knowledge from given facts by the repeated application of a predefined set of rules. Such systems used axioms of the form $A \rightarrow B$ to express the rule that "if we know that A (the antecedent) holds, then this implies that also B (the consequence) must hold". Thereby, each axiom might include several antecedents and several consequences. New facts can be concluded by chaining matching axioms. As it turned out, such pure logic-based reasoning systems are impractical to apply in most real-world applications for three main reasons:

- **Laziness:** In order to correctly represent a complex domain, a potentially huge number of axioms needs to be listed. Each axiom might

include a large number of antecedents and consequences. Despite the huge amount of work it might take to write all these axioms down, it might also be hard to practically apply the resulting set of rules.

- **Theoretical Ignorance:** Often there exists no complete theory about a domain. In this case, we simply cannot specify a complete list of axioms. Imagine the roll of a dice: There exists a logic interconnection between the resulting number of pips and the initial position of the dice, the used force, the material of the table, the air resistance and shape of the dice, et cetera. However, the detailed logical dependencies are unknown and thus cannot be specified in the form of axioms.

- **Practical Ignorance:** Even if we know all the rules, we may be uncertain about whether certain antecedents hold or not. In the dice example, we might not be able to measure the exact force that was used to roll the dice or the exact air resistance.

In many cases the available knowledge at best justifies a certain *degree of belief* in facts and causal dependencies. In order to reason about degrees of belief, we use *probability theory*, which allows us to express our degree of belief through real-numbered numerical values between 0 and 1. Moreover, probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance.

4.2 Probability Theory

In this section we introduce the basic probability notation that will be used throughout the rest of this thesis. The notation distinguishes between *prior probabilities*, which apply before any evidence is obtained, and *conditional probabilities*, which explicitly account for available evidence.

Prior Probability

We will use the notation $P(A)$ for the *unconditional* or *prior probability* that proposition A is true. If *Earthquake* denotes the proposition that at some particular point in time there is an earthquake, then

$$P(Earthquake) = 0.002$$

means, that in the absence of any other information, we assign a probability of 0.002 (a 0.2% chance) to the event of an earthquake.

Instead of talking about concrete events, propositions can also include so-called *random variables*, which can be either *discrete* or *continuous*. Discrete variables can only take on a finite set of values, while continuous variables can take on infinitely many values, i.e. all numbers in $\mathbb{R}$. In the context of this thesis, we only consider discrete random variables. Each discrete random variable X has a *domain* $< x_1, \ldots, x_n >$ of possible values that it can take on. As an example, the domain of a random variable $Weather$ could be $< rain, sunny, cloudy >$. In this case, $\mathbf{P}(Weather)$ denotes a vector of values for the probabilities of each individual state of the weather, for instance

$$\mathbf{P}(Weather) =< 0.25, 0.15, 0.6 >$$

The probability of a single value x_i of some random variable X then is written as $P(X = x_i)$. We can also understand proposition symbols as Boolean random variables. In this case, $P(X)$ is a shorthand for $P(X = true)$.

Sometimes it is useful to talk about all probabilities that are associated with values of a certain random variable at once. We call a complete assignment of probabilities to values for some random variable X a *probability distribution* over X. In this sense, the vector returned by $\mathbf{P}(Weather)$ is a probability distribution over $Weather$. Such a probability assignment might also be given in form of a function. For a discrete random variable X such a function is called a *probability mass function* over X:

Definition 4.2.1. A **probability mass function** over a discrete random variable X is a function $p_X : dom(X) \mapsto [0, 1]$[1], where

- $dom(X)$ is the domain of X
- $p_X(x)$ returns the probability of proposition $X = x$
- $\sum_{x \in dom(X)} p_X(x) = 1$

The infinite **set of all probability mass functions** of a random variable X is denoted P_X.

The probability mass function over random variable X for a known probability distribution $\mathbf{P}(X)$ is given by $p_X(x) = P(X = x)$.

[1] The interval $[0, 1]$ corresponds to the real numbers $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ in set notation.

Conditional Probability

The case of prior probability introduced above describes our belief in something in the absence of any other information. If we want to express our degree of belief that a certain proposition A holds if we already know that some other proposition B holds (and no other relevant information is available), then we denote this probability $P(A|B)$. Such a probability is called *conditional probability* of A given that B holds. In our earthquake example, the event of an earthquake becomes much more likely if the ground starts trembling. If we belief with probability 0.9 that an earthquake is happening when we the ground trembles and no other information is yet available, we express this through the conditional probability

$$P(Earthquake|Tremble) = 0.9$$

With the definition of conditional probability available, one can think of the prior probability $P(A)$ as a special case of conditional probability, where the probability $P(A) = P(A|\ )$ is conditional on no evidence.

Axioms of Probability

The following axioms are required to properly define the semantics of probabilistic statements for logical connectives. The following set of three axioms is sufficient for this purpose:

1. $0 \leq P(A) \leq 1$
2. $P(true) = 1$ and $P(false) = 0$
3. $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

The first two axioms define the probability scale: All probabilities are required to be between 0 and 1, and necessarily true propositions have probability 1 and vice versa. The third axiom defines the semantics of probabilities with respect to $\wedge$ and $\vee$. All other properties can be derived from these three axioms. The property $P(A) = 1 - P(\neg A)$ for instance can be derived from axioms 2 and 3 with the help of logical equivalence and algebra.

4.3 Bayesian Networks

Bayesian networks allow for the compact and natural representation of probabilistic knowledge. They are also called *belief* or *knowledge networks*. Thanks

to the existence of a clear graphical representation and the possibility to exploit existing conditional independencies for a factorized representation of conditional probabilities, Bayesian networks are well suited for the construction, representation, and evaluation of probabilistic knowledge models.

Representation and Interpretation

Bayesian networks are probabilistic models that represent conditional dependencies between sets of random variables. They can be visualized as directed acyclic graphs[2], where nodes correspond to random variables and directed edges correspond to conditional dependencies (accordingly, missing edges correspond to conditional independencies). Node have *conditional probability tables* (CPTs) associated which for each node define the probabilities of variable values with respect to the values of the parent node's in the graph. All CPTs together constitute a factorized representation of the model's global conditional probability table over all random variables.

Figure 4.1 shows a simple sample Bayesian network with Boolean random variables (the example is adopted from [RN09]). The network describes the situation of a house that is equipped with an alarm system and two neighbors, which might call the house's owner if they hear an alarm. An alarm might be triggered by two independent events: Either in the case of burglary, or as false alarm in the case of an earthquake. Each of these five events *Burglary, Earthquake, Alarm, John Calls,* and *Marry Calls* is represented by a Boolean random variable, which is assumed to have the value *true* if the associated event has occurred, and *false* if it has not occurred.

Arrows between nodes represent conditional dependencies. The reasons for burglary and earthquakes are not considered by this model, thus both events are assumed to happen independently by chance and accordingly do not have incoming edges. Both events might cause an alarm to go off, thus edges exist from *Burglary* to *Alarm* and from *Earthquake* to *Alarm*. If the alarm goes off, the two neighbors might call the owner of the house, which is represented by the edges from *Alarm* to *John Calls* and *Marry Calls*.

The existing conditional dependencies may be subject to uncertainty. A clever burglar might trick the alarm system, an earthquake might be too weak, or the alarm system might malfunction and cause false alarms. Also John and Mary might miss an alarm, or might call for other reasons. Bayesian networks account for such uncertainties by specifying dependencies in the form of conditional probabilities. In the example in Figure 4.1, conditional probability tables (CPT) are drawn next to each node.

[2]A directed graph is acyclic if there is no directed path $v_1 \to \cdots \to v_n$ so that $v_1 = v_n$.

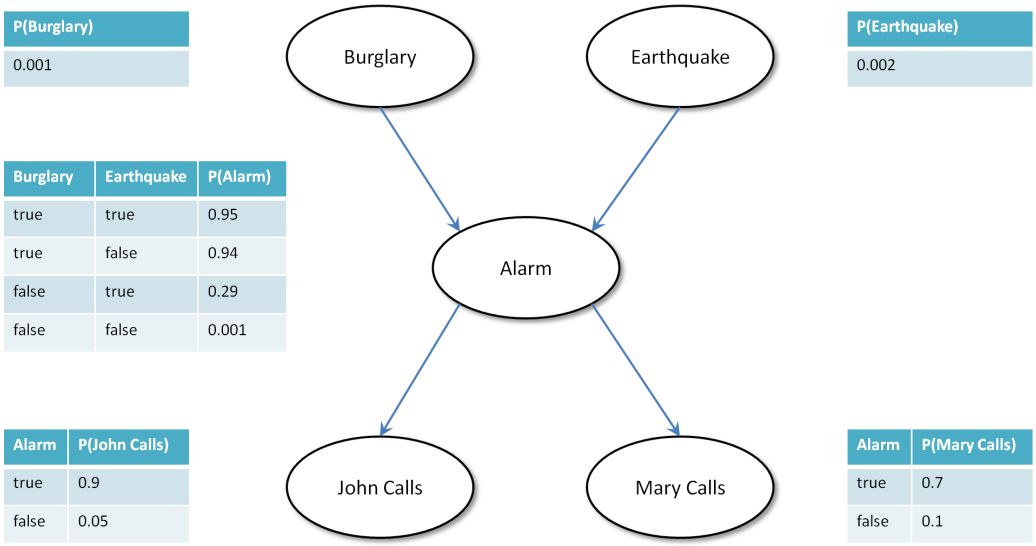


Figure 4.1: Example of a Bayesian network with Boolean random variables. Nodes represent variables, directed edges represent conditional dependencies.

For reasons of simplicity, the CPTs only contain the probabilities for an event taking place. As all variables are Boolean, we can simply compute the probability that some event X is not taking place as $P(\neg X) = 1 - P(X)$.

Probabilistic Inference

Bayesian networks define a complete but factorized conditional probability distribution over the contained random variables. We can thus use the network to answer any possible query on probabilities or probability distributions. Next we describe different kinds of such queries.

The first kind of possible query is interested in the probability of a known and complete value assignment. The general formula for the computation of such probabilities is

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | parents(X_i)) \tag{4.1}$$

where X_i is the node which belongs to value x_i, and $parents(X)$ is the set of direct parents of node X in the given network. As the states of all variables in the network are known, we can directly read the required probabilities $P(\ldots | \ldots)$ from the CPTs given as part of the Bayesian network definition.

In the kind of queries described above the true values of *all* random variables in the Bayesian network are known. Next we consider kinds of queries in which only the values of *some* of the random variables are known. We call the set of variables with known values the set of *evidence variables*. The set of nodes whose values we are interested in is called the set of *query nodes*. The goal then is to calculate the conditional probability $P(query|evidence)$. This kind of inference is called *Bayesian inference*. In their book [RN09], Russel and Norvig distinguish four different classes of Bayesian inference:

- **Causal Inference:** *From causes to effects*

  In the case of causal inference, we want to know the values of some nodes which are conditionally dependent on nodes with known values (are direct or indirect children). In our example, we might want to know the conditional probability that John calls in case there is a burglar, which is $P(JohnCalls|Burglary) = 0.849$.

- **Diagnostic Inference:** *From effects to causes*

  The case of diagnostic inference is opposite to the case of causal inference. Here, we know the value of certain nodes and want to know the values of some nodes on which the known nodes depend (are direct or indirect parents). In our example, we might want to know the conditional probability that there is a burglary if John calls, which is $P(Burglary|JohnCalls) = 0.016$.

- **Intercausal Inference:** *Between causes of a common effect*

  In intercausal inference, evidence and query nodes are not directly dependent on each other, but have at least one common child. As evidence nodes together influence the value of common child nodes, an effect called *explaining away* occurs. Assume that we are interested in a burglary and know about an alarm, so we have $P(Burglary|Alarm) = 0.374$. If we now observe an earthquake, we have $P(Burglary|Alarm \wedge Earthquake) = 0.003$. As a result, *Burglary* is explained away by *Earthquake*, although both are conditionally independent.

- **Mixed Inference:** *Combining two or more of the above*

  In the case of mixed inference arbitrary nodes might be used as evidence and query nodes. To answer such queries, we have to apply combinations of at least two of the aforementioned inferences. In our example, we might be interested in the probability that an alarm

was triggered if John calls and there is no earthquake. By applying a combination of causal inference and diagnostic inference we get $P(Alarm|JohnCalls \wedge \neg Earthquake) = 0.034$.

All four kinds of probabilistic inference can be performed with a single algorithm that exploits causal independencies that exist between nodes. The basic idea is to propagate causal evidence towards child nodes, while propagating diagnostic evidence towards parent nodes. For the complete algorithm the interested reader is pointed to [RN09].

Dynamic Bayesian Networks and Markov Chains

A *dynamic Bayesian network* (DBN) represents discrete sequences of variables, such as time-series or sequences of symbols. The evolution of a set of variables is described by a sequence of Bayesian networks with uniform structure, where each network represents one slice of the sequence. A roll-up function describes how evidence is propagated between adjacent networks.

Figure 4.2 shows two time slices of a simple DBN which models a blinking light. The state of the light at the beginning and the end of each time slice is represented by two Boolean variables *LightWasOn* and *LightIsOn*. The value of the latter is the inverse of the value of the former variable (cf. CPT of *LightIsOn*). The state at the beginning of a time slice equals the state of the light at the end of the previous time slice, hence the roll-up function is the identity function as reflected by the CPT of variable *LightWasOn*[3].

It should be mentioned that the same sequence can be modeled by a DBN with only a single node and an inverse roll-up function. We introduced the second node to better illustrate the example.

The sequence of light states resulting from our example has the property, that the light state in time slice $t+1$ only depends on the light state in time slice t. In other words, the description of the present state fully captures all information that influences the future evolution of the process. A stochastic process with this property is called a *Markov chain*. Markov chains always can be represented as a DBN. We exploit this property in chapter 6 to represent the agent's plan selection and execution process as a DBN.

Probabilistic inference can be performed in DBNs similar to probabilistic inference in regular Bayesian networks. As an alternative solution, Brandherm and Jameson describe a method to translate DBNs into multivariate polynomials, and to compile these polynomials into native code which allows for an efficient constant-space evaluation of the represented DBN [BJ04].

[3] In order to better distinguish between variables in adjacent time slices we append a dash to all variable names in the successor time slice.

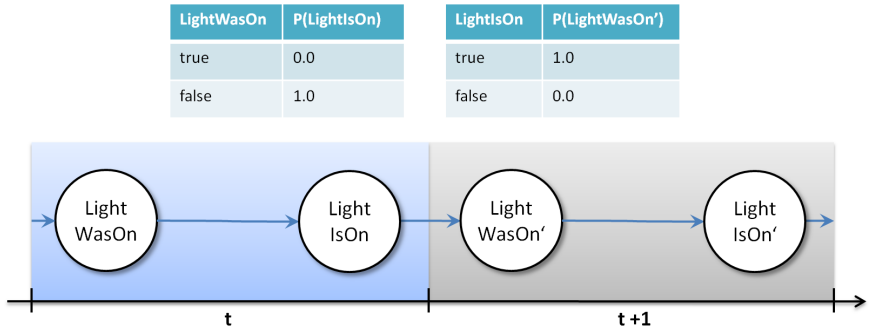


Figure 4.2: Dynamic Bayesian Network modeling a blinking light.

4.4 Decision Theory

The area of *decision theory* investigates questions related to rational decision making under uncertainty. This includes how the utility of decisions can be expressed and calculated, how optimal decisions can be made, and how uncertainty about the current situation might influence a decision situation.

Decision Problem

A *decision problem* describes a situation in which an agent has to choose between competing options. The outcome of each option depends on variables that are not under the agent's control. To express preferences for individual outcomes, we assign a numerical rating (called *utility*) to each combination of option and variable value. Utility values can express preferences relative to material goods like money or scoops of ice cream, but can also relate to immaterial things like happiness or good karma (however this might be measured in the concrete case). We assume that higher utilities are always preferred over lower utilities[4].

An everyday decision problem is the *umbrella decision problem*, which is the question whether or not to take an umbrella along when leaving home in the morning. The outcome of this decision depends on the variable *Weather*, which we assume can take on one of the two values *rain* and *sun*. Our preference for certain outcomes is determined by the competing goals of avoiding the burden of carrying an umbrella and staying dry. Assume that the utility of staying dry is equivalent to earning four dollars, and that the burden of carrying an umbrella is equivalent to losing one dollar (in the following

[4]This property in practice does not necessarily hold for all preference measures, e.g. there might be an upper bound on the amount of ice cream one may want to eat.

we will omit the unit). Now we can calculate the outcome utilities for each possible combination of choice and weather condition (see Table 4.1).

	Weather = rain	**Weather = sun**
Take Umbrella	I stay dry but carry the umbrella. (3)	I stay dry but carry the umbrella. (3)
Do Not Take Umbrella	I get wet. (0)	I stay dry. (4)

Table 4.1: Possible outcomes of the umbrella decision problem with resulting utility values (in parentheses).

From this table, we can easily look up what is the best option if we already know the actual weather. We simply have to compare the preference values in the according column of the table, and chose the option with the highest utility value. In the case of rain, we prefer taking an umbrella (utility 3) to not taking an umbrella (utility 0). In the case of sun we prefer taking no umbrella (utility 4) to taking an umbrella (utility 3).

Decision problems can be visualized as *influence diagrams*, which are directed graphs where round nodes represent random variables, rectangular nodes represent decision situations, rhombic nodes represent utilities, and arcs represent conditional dependencies. Figure 4.3 shows two influence diagrams for the umbrella decision problem. For now we only consider the left diagram: The *Weather* node represents the actual weather condition. The arc from *Weather* to *Umbrella* shows that the value of the *Weather* node is known to the decision maker. The arcs from *Weather* and *Umbrella* to *Utility* indicate, that the resulting outcome utility of the decision problem depends on the values of the *Weather* variable and the chosen *Umbrella* option.

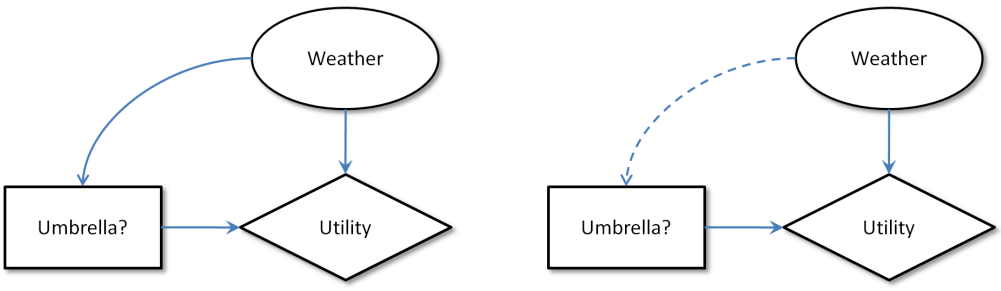


Figure 4.3: Influence diagram of the umbrella decision problem with complete (left) and incomplete information (right) about the actual weather.

If we understand the set of all variables which influence the utility of a decision as the *decision situation*, we can defined a decision problem as:

Definition 4.4.1. A **decision problem** is a triple $(\mathcal{B}, \mathcal{V}, u)$, where

- $\mathcal{B}$ is a finite *set of options*
- $\mathcal{V}$ is a finite *set of situations*
- $u : \mathcal{B} \times \mathcal{V} \mapsto \mathbb{R}$ is the *utility function*, where $u(\beta, v)$ returns the utility of choosing option β in situation v

Making Optimal Decisions Given Incomplete Information

The previous section showed how to solve decision problems if complete information about relevant variables is available. Now we show how to decide for the best option in the case of incomplete information. Incomplete information about the state of a random variable usually is represented by a probability mass function over the set of possible values.

In the case of incomplete information we have to choose an option which provides the best trade-off between the resulting utility in case we have "accidentally" chosen the right/wrong option. For a fixed option, we can compute the so-called *expected utility* as the average of possible outcome utilities weighted by the probability of the according outcome. Probability values are given by the probability mass function that represents our incomplete information about the actual situation. Formally we define expected utility as:

Definition 4.4.2. The **expected utility** of choosing option $\beta \in \mathcal{B}$ in a decision problem $DP = (\mathcal{B}, \mathcal{V}, u)$ given knowledge $p_{\mathcal{V}}$ about the current situation is defined as a function $EU_{DP} : \mathcal{B} \times P_{\mathcal{V}} \mapsto \mathbb{R}$ with

$$EU_{DP}(\beta, p_{\mathcal{V}}) = \sum_{v \in \mathcal{V}} p_{\mathcal{V}}(v) u(\beta, v)$$

Table 4.2 shows how to compute the expected utilities in our umbrella example if we know from the weather forecast that the probability of rain

is 0.2, and the probability of sun is 0.8. The last table column shows the resulting expected utility values. In order to decide for the most promising option, we chose the row with maximum expected utility and find a slight preference for not taking an umbrella (utility 3.2 over 3.0).

	Weather = rain $p_{Weather}(rain) = 0.2$	**Weather = sun** $p_{Weather}(sun) = 0.8$	**Expected Utility**
Take Umbrella	I stay dry but carry the umbrella. (3)	I stay dry but carry the umbrella. (3)	$0.2 \cdot 3 + 0.8 \cdot 3 = 3.0$
Do Not Take Umbrella	I get wet. (0)	I stay dry. (4)	$0.2 \cdot 0 + 0.8 \cdot 4 = 3.2$

Table 4.2: Expected utilities in the umbrella decision problem given incomplete information.

Like in the case of complete information, an influence diagram can visualize the existing dependencies in a decision problem given incomplete information (see Figure 4.3, right). The availability of incomplete information here is represented by the dashed line from the *Weather* node to the *Umbrella* node.

Note, that the case of perfect information is just a special case of the more general case of imperfect information. Now that we have defined the expected utility of an individual option, we can use it to formally define the solution of a decision problem. Decision theory assumes that a rational decision maker always chooses the option which yields the highest expected utility. Recall that we assumed earlier in this section that higher utility values are always preferred over lower utility values.

The option which provides us with the highest expected utility is called the *solution of the decision problem*. If several options with the same maximum expected utility exist, then all of these options are valid solutions of the considered decision problem:

Definition 4.4.3. The **solution** of a decision problem $DP = (\mathcal{B}, \mathcal{V}, u)$ given knowledge expressed by $p_{\mathcal{V}}$ about the current situation is an option $\beta^* \in \mathcal{B}$ with

$$EU_{DP}(\beta^*, p_{\mathcal{V}}) = \max_{\beta \in \mathcal{B}} EU_{DP}(\beta, p_{\mathcal{V}})$$

If we know the solution of a decision problem, we can use the expected utility function to calculate the expected utility of this option (in fact, this utility has most probably already been computed in order to solve the decision problem). The expected utility of the best option is called the decision problem's *maximum expected utility*, and provides the expected utility of the overall decision problem. Maximum expected utility is formally defined as:

Definition 4.4.4. The **maximum expected utility** of a decision problem $DP = (\mathcal{B}, \mathcal{V}, u)$ with solution β^* given knowledge expressed by $p_{\mathcal{V}}$ about the current situation is defined as a function $\widehat{EU}_{DP} : P_{\mathcal{V}} \mapsto \mathbb{R}$ with

$$\widehat{EU}_{DP}(p_{\mathcal{V}}) = EU_{DP}(\beta^*, p_{\mathcal{V}})$$

4.5 Summary

In this chapter we introduced the general theoretical frameworks that we apply in the rest of this thesis. We motivated why it is reasonable to consider uncertainty in the reasoning of real-world systems. Our approach uses probability theory as the main framework for reasoning under uncertainty. We presented the concept of (dynamic) Bayesian networks, which allow for the representation of probabilistic knowledge. We use Bayesian networks in chapter 6 to model the observed agent's plan selection and execution process as well as the sensors in the instrumented environment. We introduced decision theory and decision diagrams as means of reasoning about decisions under uncertainty and resulting utilities, which provides the theoretic foundation for the definition of our utility model for observation information in chapter 7.

Anyone who says that they can contemplate quantum mechanics without becoming dizzy has not understood the concept in the least.

Nils Bohr (1885–1962)

5 Related Work

In this chapter we review and discuss other research that is related to the topic of this thesis. In section 5.1 we start by giving examples of existing *instrumented environments*. Section 5.2 describes previous research in the area of *plan recognition* itself. Related work in the area of *sensor selection* then is presented in section 5.3. In section 5.4 we describe related work that addresses *decision models* and reasoning about different properties of such models.

Section 5.5 concludes this chapter with a *summary* of insights that we learned from our study of related work. We explain the choices we made for our own plan recognition approach, motivate why we follow a utility-based approach for sensor selection, and discuss the feasibility of approaches for reasoning about knowledge models for the problem of computing the utility of observation information.

5.1 Instrumented Environments

In this section we describe three different existing instrumented environments. We introduce their general idea and setup and point out how plan recognition can be used to support agents in the presented environments.

5.1.1 Smart Factory

The *Smart Factory* is a research, demonstration, and evaluation center for technologies that might help planning, assembling, running, maintaining, and reconfiguring future manufacturing facilities. It is based on the Smart Factory initiative [Sma09] and operated by the German Research Center for Artificial Intelligence (DFKI) in Kaiserslautern. The envisioned factory of the future has four main qualities:

- **Flexibility:** The factory of the future consists of small, interchangeable function units, which allow for an easy and economical construction of customized production lines. The modular design furthermore helps to easily adapt existing production lines to changing requirements.

- **Interoperability:** Through standardization of mechanical connectors, electrical systems, networking protocols, process control, data formats, and IT interfaces, function units of different manufactures can be seamlessly integrated and efficiently cooperate in the factory of the future.

- **Self-Organizing:** By exploiting context knowledge, related function units can autonomously negotiate configuration parameters at assembly time, or can intelligently optimize production processes at runtime, e.g. by rescheduling jobs that make use of a particular function unit that is currently busy or under maintenance.

- **User-Centeredness:** The factory of the future provides its users with intelligent and context-aware support in designing, operating, adapting, and maintaining production lines. Again, context knowledge is used to provide relevant information or to allow for the setting of function unit parameters via a standardized interface on the user's mobile device [BMG+09].

In order to improve flexibility and modularity, function units in the *Smart Factory* use wireless technologies like WiFi, Bluetooth, ZigBee, and UMTS to communication with each other and with the central factory control system. Hence, the *Smart Factory* becomes a wireless factory, where physical connections between function units are minimized to a single connection for power, pressurized air, and connections used to transport the produced goods.

An important part of the *Smart Factory* vision is to establish a clear separation between production hardware and software layer, for instance by representing the control logic of individual function units as a collection of web services. Such a separation allows for abstracting from the used hardware and to simulate and test whole production processes in a virtual counterpart of the physical production environment. Figure 5.1 shows selected parts of the existing physical production line, as well as the corresponding virtual 3D model that is used for simulation and control.

In the *Smart Factory* environment, multiple sensors observe the factory's operation and the user's actions. Besides specific sensors and actuators that are part of the individual function units, some general purpose instrumentation is applied. Here, RFID (see subsection 2.3.2) plays a major role: Every

Figure 5.1: 3D model of the *Smart Factory* production environment (top) and its physical counterpart (bottom). Source: [Sma09].

manufactured item is equipped with an RFID tag, and RFID antennas are located at the beginning and at the end of each function unit. Thus, the system can track the current location and production progress of every item.

In order to track items and users in the vicinity of the production line, several indoor positioning systems are applied. One of the systems uses passive RFID transponders that are embedded in the production facility's floor forming a grid. Each transponder contains its absolute position. Movable objects like an *instrumented workshop cart* can detect these transponders (and thus infer their current position) via RFID antennas mounted to the bottom side. Further positioning approaches that have been applied and tested include ultra wide band (UWB) and ultrasound technologies [SHKF09]. Based on location information and knowledge about near-by system components and function units, location based services can be realized that provide context-dependent assistance, e.g. by showing relevant operation parameters of close-by machines to workers during maintenance and repair.

Information from general purpose sensors as well as from specific function unit sensors can be used to perform plan recognition in the *Smart Factory* environment. In addition to sensor observations provided by physical sensors, plan recognition might also exploit knowledge about recent changes to configuration parameters or process control in order to predict a user's intentions. This for instance is useful to infer, which kind of maintenance a technician intends to perform on a certain component or function unit. Upon this information, the technician might be provided with proactive assistance, for instance by utilizing the 3D model of the production line to visualize relevant components and to proactively generate animations that demonstrate the actions that have to be performed by the user.

5.1.2 Bremen Ambient Assisted Living Lab

The *Bremen Ambient Assisted Living Lab (BAAL)* [Ger09a] is an instrumented environment that supports elderly and people with physical or cognitive impairments in their everyday living. It has a size of $60\,\mathrm{m}^2$ and allows investigating all conditions of the basic living of two persons. It is operated by DFKI and located in Bremen.

One of the main focuses of the ambient assisted living laboratory is to support the accessibility of the home environment to physically handicapped people that live together with their (possibly non-handicapped) partners. For this purpose the BAAL is equipped with systems that allow for the dynamic and automated adaptation of the environment to the user's physical limitations. Examples are a user-adaptable kitchenette, which can be electrically raised and lowered (see Figure 5.2, bottom), and a bed mattress which can be adjusted to individual positions. Automated sliding doors are another example of instrumentation that is applied in the BAAL to help impaired users living autonomously in their home environment for as long as possible.

A second focus of the ambient assisted living environment is mobility assistance, which is provided through an *intelligent wheelchair* and an *intelligent walker* (see Figure 5.2, top). Both, the wheelchair and the walker, are equipped with sensors and actuators to assist safe driving (braking, automatic obstacle avoidance) as well as navigation to known destinations.

A major challenge is to ensure secure and smooth interoperability of the involved systems, and to provide an intuitive, multimodal user interface that allows impaired users to control the environment with minimum cognitive and physical effort. A typical use case looks as follows (adapted from [Ger09a]):

> John, a paraplegic user is sitting at the desk in his intelligent wheelchair. He gets hungry and tells the wheelchair via voice

Figure 5.2: The *Bremen Ambient Assisted Living Lab* supports elderly and impaired people in their everyday living, e.g. by autonomous navigation of a smart wheelchair and walker (top), or by proactively adjusting the height of kitchen shelves to the user's needs (bottom). Source: [Ger09a].

control that he wants to have a pizza. As a result, the wheelchair drives him to the kitchen, while sliding doors open and close automatically (see Figure 5.2, top right).

Once John is in the kitchen, the kitchenette moves to a height that allows the wheelchair to securely drive under it. In addition, the kitchen cupboard moves downward so that John can access everything inside (see Figure 5.2, bottom left). John gets a plate and asks "Where is the pizza"? Now, the wheelchair drives John to the fridge, which he opens. The compartment that contains the pizza is blinking, so that John can easily locate and grab the pizza. Next, the wheelchair drives him to the microwave oven that is also moving down to be accessible. After the pizza is ready, John asks the wheelchair to drive him to the table. There, the light is automatically switched on, while the light in the kitchenette is switched off.

To realize such applications, the environment is equipped with various intelligent devices and objects, which can report about their current state and use. Examples are the intelligent wheelchair, which is equipped with laser range scanners for positioning, or the adjustable kitchen, which can report about its current configuration. Together with sensors like motion sensors or RFID-tagged objects, it is possible to closely follow the user's actions in the environment.

This information can be used to perform plan recognition in the BAAL environment. Here, plan recognition could serve two main purposes: On the one hand, it can anticipate the user's plan and execute actions on behalf of him, like lowering the microwave if the system expects that the user is preparing a pizza (even if the user did not explicitly express this desire). This kind of support is particular interesting for users with physical handicaps. The second kind of support is particular interesting for users suffering from dementia. Here, plan recognition might be used to recognize the start of a plan, and then watch for abnormal behavior, which might indicate that the user is distracted from following the started plan. In this case, the user can be reminded to complete the plan, or the system itself might execute missing steps (like turning off the stove after cooking).

5.1.3 Innovative Retail Laboratory

The *Innovative Retail Laboratory* (*IRL*) [Ger09b] is an application-oriented research laboratory of DFKI and is installed at the head office of the German chain store GLOBUS in St. Wendel. With a total size of over $500\,m^2$, the IRL provides a unique environment for intelligent shopping-related applications with the goal of researching novel ways of supporting customers and shop owners. Topics of interest include virtual assistants responsible for matters of dieting and allergies, personalized cross-selling and up-selling, smart items with digital product memories, indoor positioning and navigation, as well as novel logistics concepts. Implemented prototypes include personalized and multi modal shopping assistants [WS04], anthropomorphic products which talk to the customers [SKS07], as well as intelligent shopping carts which plan and show the way through the store according to the customer's shopping list.

A typical scenario that is supported by the IRL environment is the following: At home, the user creates a shopping list at her smart fridge (see Figure 5.3, top left). The shopping list can then be transferred to the user's mobile device, and taken to the store. Here, the user logs into an instrumented shopping cart via a finger print reader (see Figure 5.3, top right). On success, the shopping list is transferred from the user's mobile device to

the shopping cart, which calculates the optimal path through the store[1] and provides the user with navigation information.

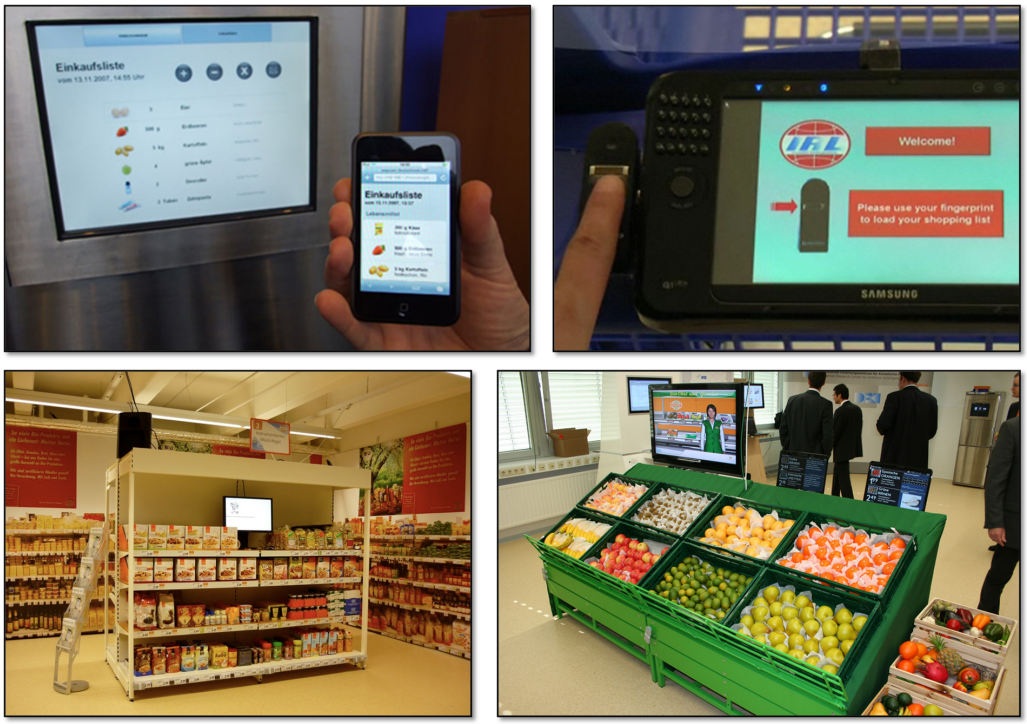


Figure 5.3: The *Innovative Retail Laboratory* provides a unique environment for intelligent shopping applications like the creation of shopping lists at home (top left), their use with an intelligent shopping cart (top right), instrumented shelves (bottom left), and virtual shopping assistants (bottom right).

The current position of the shopping cart is detected via passive RFID transponders embedded into the shop floor and attached to shelves, and a mobile RFID antenna mounted to an instrumented shopping cart. A second RFID antenna is mounted to the cart's basked and is used to detect the set of RFID-tagged products that the customer has chosen so far.

Information about selected products is used by the shopping cart to tick items off the shopping list, and to calculate the total price of products selected so far. In addition to the shopping carts, all product shelves in the store are equipped with RFID antennas (see subsection 2.3.2), which track

[1]The optimal path proposed by the shopping cart might consider additional aspects besides the products chosen by the customer, e.g. current waiting times at meat or cheese counters, or advertisement areas for special sales.

the customer's interactions with the RFID-tagged products. Various displays mounted to shelves provide additional product information or product comparisons, based on the products recently taken from the shelf (see Figure 5.3, bottom left).

Expensive products like bottles of champagne or high quality wines are additionally equipped with wireless sensor nodes (see subsection 2.3.1) to gather further information about the customer's interactions with the products [MSD08]. Acceleration sensors for instance are used to sense the bottle's orientation, and allow to conclude at which side of the box (and thus at which information) the customer is currently looking. Complex information might be presented to the customer via virtual life-like character sales assistants, which serve as domain experts, e.g. at an instrumented fruit and vegetable stall (see Figure 5.3, bottom right).

The idea of applying plan recognition in instrumented real-world shopping environments has been introduced by the author of this thesis in [Sch04]. The shopping domain has some characteristics that make it especially well suited for the application of plan recognition techniques. On the one hand, the actions of a user in a shopping environment mostly fall into one of two categories: (1) Navigating in the shop and (2) interacting with products. The IRL demonstrates how both kinds of actions can be observed in realistic environments with existing technology. Further information might be available to the plan recognition system, e.g. if the customer shares her digital shopping list with the shop (see example above).

Another advantage of the shopping domain is the common understanding of possible user plans and their limited number of realizations. Most plans fall into one of two general classes: The first is gathering product information and the second one is buying a set of (partially) related products. Upon recognition of the customers intentions, the shop may provide her with context-dependent assistance, i.e. by recommending similar or better suited products or add-ons (proactive up-selling/cross-selling). The user might also be reminded about products, that she eventually forgot to add to her shopping list, but which are required for the assumed plan (e.g. in case a customer buys everything needed for a barbecue, but is not buying any meat).

5.2 Plan Recognition

In this section we discuss related work in the area of plan and activity recognition. We review existing recognition approaches that make use of a broad range of different theoretical frameworks, present typical application areas, and shortly discuss the computational complexity of plan recognition.

5.2.1 Explanation-Based Understanding

The first system that used plan-recognition-like techniques was PAM [Wil78, Wil83], which was developed by Wilensky to solve the problem of story understanding. In order to understand a story, the reader (in this case the computer system) has to infer the intentions and goals of the story characters. In his work, Wilensky introduces the concept of *explanation-based understanding*. The goal of explanation-based understanding is to build a set of explanation chains that connect observed actions with higher-order goals. PAM tries to link newly observed actions to existing explanation chains whenever possible. If no matching chain is found, PAM starts a new chain to explain the observed action. If there exists more than one explanation chain at the end of the story, PAM chooses the shortest chain available as the final explanation.

PAM has been criticized for two reasons: Firstly, the pure length of an explanation chain in general does not seem to be a good indicator for the plausibility of an explanation. Secondly, explanation chains are never revised. If an action was incorrectly attached to an existing chain, this interpretation cannot be changed at a later point in time if new evidence becomes available. As a result, PAM cannot correct potentially wrong interpretations.

5.2.2 Logic-Based Plan Recognition

One of the first general purpose plan recognition approaches was proposed by Kautz in [Kau91]. Kautz used *propositional logic* to formulate plan recognition as a problem of theorem proving. Starting from a set of axioms which resemble the plan library and the observed actions, the problem of plan recognition can be solved with "standard" theorem proving software. Kautz further introduced *hierarchical plan libraries*, which allow to reason about abstract classes of plans whenever detailed knowledge is not (yet) available to the plan recognition system. Thus, a kitchen might recognize that the user is preparing a pasta dish even if it is unclear whether it will be spaghetti carbonara or spaghetti vongole.

An approach for logic-based distributed multi-level plan recognition is proposed by Hecking [Hec93]. Here, *multi-level plan recognition* means that a hierarchical set of plan recognizers exist, where higher-level plan recognizers perform plan recognition based on the results of lower-level plan recognizers. Furthermore, Hecking's approach considers *distributed plan recognition* as cooperative plan recognition of multiple plan recognizers where each single plan recognizers only knows about parts of a single agent's actions or observes only a single agent from a group of collaborating agents. By fus-

ing partial recognition results from multiple recognizers, one finally can infer the overall goal of the observed agent or group of agents. For this purpose, Hecking proposes the logical model $\mathcal{MOD} - \mathcal{PE}$, which uses *belief operators* to represent the individual plan recognizers' beliefs about the agent's plans. The process of plan recognition then is performed through logical deduction in sequent calculus for modal logic, which Hecking implements through a meta-interpreter in PROLOG.

The logic-based foundation of the plan recognition problem allows for the accurate and potentially fine-grained representation of the causal relations between actions, plans, and observations. The level of detail is only limited by the expressiveness of the underlying logical calculus. At the same time, a big drawback of pure logic-based plan recognition approaches is their inability to deal with the uncertainty inherent to observations and plan hypotheses. Although it is possible to implicitly express preferences for a given set of plans through the careful and manual addition of further axioms – a process called *circumscription* – it is unclear how this approach can be automatically applied in the general case.

5.2.3 Probabilistic Plan Recognition

To overcome the above mentioned problems of logic-based approaches, plan recognition systems were proposed that made use of techniques for reasoning under uncertainty. Such systems do not generate a single explanation for a series of observations, but instead return a set of possible explanation alternatives together with a rating of the likeliness of each explanation alternative.

Charniak and Goldman use *Bayesian networks* [Pea85, Pea88] to model the conditional probabilities between plans and (sub)actions [CG93]. They provide a set of rules which allow transforming plan libraries into Bayesian networks with a special structure, so-called *plan recognition networks*. As the complete transformation of whole plan libraries would result in very large and complex networks that are practically intractable, Charniak and Goldman propose a two-step approach: In the first step, potential candidate explanations are identified through a breadth-first search in the plan library. In the second step, a minimal plan recognition network is constructed only for those plans that have been identified as candidates in the first step. This network is then evaluated to compute the exact probabilities of individual plan candidates.

An extension of Bayesian networks, so-called *Probabilistic Relational Models* (PRMs, [PKMT99]), have been applied by the author of this thesis in [Sch03] to realize an object-oriented plan recognition system. The focus of

this system is plan recognition in open and dynamic domains, as it separates plan-related knowledge from distributed context knowledge about agents and movable objects. The underlying assumption is, that the general set of possible plans depends on the given environment and thus is static, while the likelihood that certain plans and actions are executed depends on the individual agent and current context and thus is dynamic. For this reason the static plan library only contains abstract models of the agent's behavior and context. At runtime, these placeholders are substituted with concrete model instances whenever additional information about the agent or some context item is available. The resulting plan recognition network is then used to compute the exact probabilities of individual plan candidates in a way that is similar to the approach of Charniak and Goldman [CG93].

A probabilistic plan recognition framework based on *Dempster-Shafer theory* [Dem68, Sha76] is proposed by Bauer [Bau94]. Bauer argues that in many application domains it is unrealistic to have complete knowledge about the underlying probabilistic model, as this would require the assessment of a large number of conditional probabilities. He proposes to overcome this problem through the application of Dempster-Shafer theory, which allows for the explicit representation of incomplete probability information. The central idea of Dempster-Shafer theory is to distribute "evidence mass" over the power set of the set of basic hypotheses. With respect to plan recognition, Bauer defines the set of basic hypotheses to be the set of observable actions. Each plan consists of a (sub)set of these actions and thus can be seen as an element from the power set of actions. If the execution of an action is observed, evidence mass is assigned to all plans that contain this action. Bauer presents an algorithm that updates the distribution of evidence mass without knowledge about the underlying conditional probabilities. The likelihood of individual plan candidates can then be assessed from the distribution of the evidence mass.

A probabilistic plan recognition approach that uses parsing techniques is proposed by Pynadath and Wellman [PW00]. The authors introduce *Probabilistic State-Dependant Grammars* (PSDG) as a representation language for plan libraries. Such grammars have a state variable that influences the probabilities that individual production rules are applied. State transitions occur with given probabilities whenever a terminal symbol is generated. The set of all plans for a given grammar then equals the set of sentences that can be generated with this grammar, and the probability of a plan is the combined probability of the involved production rules. In order to use this approach for the recognition of plans, a parse tree is generated and used to derive a Bayesian network, which then allows computing the probabilities of individual plan hypotheses.

Bui describes in [Bui03] a plan recognition approach based on *Abstract Hidden Markov Memory Models* (AHMEM, [Bui02]). AHMEMs are an extension of abstract hidden Markov models and allow the resulting policy to have internal memory which can be updated in a Markovian fashion. AHMEMs can represent a richer class of probabilistic plans than traditional Markov models. Bui derives an efficient algorithm for plan recognition in AHMEMs based on a Rao-Blackwellised particle filter [CR96] approximate inference algorithm. This algorithm is based on a dynamic Bayesian network (DBN) [Kja92] representation of the AHMEM, where special "context variable" nodes are introduced in each time slice to represent the policy's internal memory.

Due to the ability to deal with the uncertainty inherent to any plan recognition application all state-of-the-art plan recognition approaches make use of methods for probabilistic knowledge representation and reasoning. The advantages of such methods become obvious if plan recognition is applied in real-world environments, in which uncertainty about executed plans, sensor observations, and agent intentions plays an even more important role due to factors like partial observability, unavoidable observation errors, or incomplete domain knowledge.

On the downside, plan recognition approaches like those proposed by Charniak and Goldman or Bauer do not allow for the direct representation of state information and hence cannot natively consider state observations in their reasoning. Pynadath's and Wellman's probabilistic state-dependent grammars (PSDGs) use an internal state variable to control the invocation of production rules, but it remains unclear if (and how) this internal state corresponds to the state of the physical environment.

5.2.4 Activity Recognition

The general term "activity recognition" relates to a broad range of approaches which aim at *recognizing behavior and goals* of agents from sets of sensor data. The term is not uniquely defined and depending on the context is used for approaches like location estimation, event recognition, intent recognition, and sometimes also plan recognition.

In the context of the following discussion we distinguish between event recognition, activity recognition, and plan recognition on the basis of the complexity and level of abstraction of the explanations that are inferred from the collected observations. The resulting hierarchy of approaches is illustrated in Figure 5.4 and explained in the following.

The foundation for all considered behavior recognition approaches is provided by the raw data that is collected by the sensors in the environment.

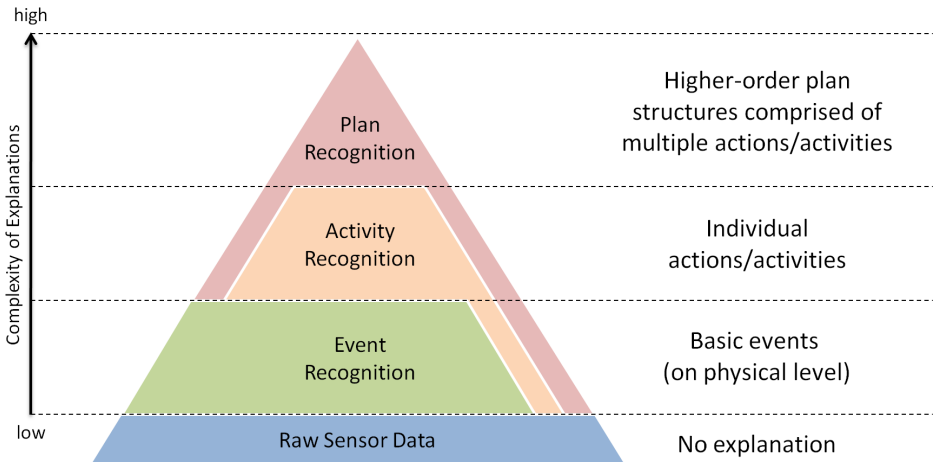


Figure 5.4: Event recognition, activity recognition, and plan recognition explain an agent's behavior at different levels of abstraction and detail.

Without any further processing this data is usually not "interpretable" by a computer system, as it lacks a well-defined semantic. Hence this data has to be explained and transferred into information with well-defined semantics through (a cascade of) abstraction processes.

On the lowest level, *event recognition* tries to infer basic events from raw sensor data. Events often describe derived state changes on the physical level. An example for a system performing event recognition is the automated soccer commentator system *SOCCER* [HRS89], which analyzes physical trajectories of moving objects from sequences of real-world image data to extract events like running or passing the ball. Event recognition ranges from simple approaches like thresholding on a single sensed variable to complex approach that for instance consider the evolution of large and/or fused sets of observed variables over longer periods of time [WKY+09]. Events usually have none or only a short extension over time.

On the next higher level, *activity recognition* aims to recognize more complex agent behavior in the form of activities, which typically extend over a longer time period and may involve the occurrence of several events. Examples for possible activities in the kitchen domain are "cooking", "dish washing" or "eating". Activity recognition approaches may directly operate on raw sensor data or can use techniques for event recognition to preprocess raw observation information. Existing approaches that perform activity recognition are presented later in this section.

On the highest level *plan recognition* tries to explain an agent's behavior through higher-order plans, which usually are structures comprised of

multiple individual actions or activities. Plans always have a temporal extension. In contrast to activity recognition, plan recognition for instance can not only infer that the agent is currently performing the activity "cooking", but additionally can explain which recipe the agent is preparing, at which point in the cooking process the agent currently is, or what is needed to successfully complete the current activity of cooking. Hence, plan recognition usually provides more detailed explanations than activity recognition. Often plan recognition is performed on the basis of already recognized activities or events, but it can also be performed on raw sensor information if the applied plan recognition approach is powerful enough (the work presented in this thesis is an example for such an integrated approach). The automated soccer commentator system *SOCCER* [HRS89] that we introduce above for instances has a plan recognition component *REPLAI* that reuses information recognized by its event recognition component to interpret the current intentions of the players.

As the above discussion indicates, the concepts of event recognition, activity recognition, and plan recognition are closely related to each other, thus it is often difficult to clearly classify a given system into one of the categories. In the following we discuss some work that mostly falls into the category of activity recognition as described above.

Liao, Fox, and Kautz for instance perform *location-based* activity recognition in order to identify an agent's *significant places* from GPS data and learn to *discriminate between the activities performed at these locations*, in particular {'AtHome', 'AtWork', 'Shopping', 'DiningOut', 'Visiting', 'Others'} [LFK05]. As additional evidence Liao et al. utilize *temporal information* (time of day), *spatial information* from geographic databases (kinds of businesses in agent's proximity), and *sequential information* on activities (follow-up frequencies). Their approach is based on *Relational Markov Networks* (RMNs), which are undirected graphical models that extend the idea of conditional random fields [LMP01]. RMNs provide a relational language for describing parameter constraints through clique templates. Liao et al. present a general framework for sensor-based activity recognition and show how to efficiently learn and reason in this framework using Markov-chain Monte-Carlo algorithms.

A second example of a system performing activity recognition is S-SEER, a multimodal office activity recognition system by Oliver and Horvitz [OH04]. The authors use a cascade of Hidden Markov Models to recognize an agent's behavior in an office environment. Concretely, the system tries to recognize the six activities {'PhoneConversation', 'Presentation', 'FaceToFaceConversation', 'DistantConversation', 'OtherActivity', 'NobodyPresent'} from real-time streams of video, audio and computer (keyboard and mouse) interaction

information. As a special feature, the system employs an expected-value-of-information measure to dynamically select evidence sources in order to limit sensing and data analysis costs in a context-sensitive way (see discussion in subsection 5.3.3). This approach allows for the significant reduction of CPU time usage on the system that is running the activity recognition software with a minimal impact on recognition accuracy.

The state-aware plan recognition approach that we present in chapter 6 uses a Bayesian sensor model to link executed actions and resulting state transitions to observation data. Depending on the complexity of this sensor model and the abstraction level of the plan library, possible observations can range from low level sensor data over derived events to recognized activities. The proposed approach is flexible enough to be applied with different degrees of preprocessed observation data.

5.2.5 Expressiveness of Plan Recognition Models

The expressiveness of the applied plan recognition model determines the structure and complexity of plans that can be recognized by a plan recognition system. Expressive plan recognition models on the one hand allow for the recognition of a broad range of plans, but on the other hand increase the computational complexity of the plan recognition problem (see subsection 5.2.7). Hence existing plan recognition systems have to find a suitable trade-off between representable plan complexity and resulting recognition performance. In the following we present a selection of plan recognition approaches that provide examples of different degrees of expressiveness along various dimensions of the plan recognition space.

Recursive plans represent agent behavior where the realization of a partial plan (in form of a subplan or an abstract action) is expanded to a set of subplans or abstract actions which possibly contains the expanded subplan/action itself. An example for a recursive plan in the kitchen domain is a "Wash vegetables" plan that consists of two subplans: "Apply water and rub", and "Check if clean". If the check reveals that the vegetables are still dirty, they have to be washed again, which results in an expansion of the "Check if clean" subplan with a new instance of "Wash vegetables". Recursive plans are typically supported by plan recognition approaches like [CG93, PW00] which define the plan library in form of expansions rules.

Multi-party plans describe the behavior of a group of agents which interact and/or cooperate to follow a common plan and reach a common goal. Plan recognition approaches that allow for the recognition of multi-party plans have been developed and applied in the domains of adversarial military and security surveillance [RM94], for the recognition and prediction of

team behavior in sporting events like soccer matches of human players [RS92] or between teams of robots [HV99]. Multi-party plans are easy to recognize by most plan recognition approaches if the subtasks in such plans are explicitly assigned to distinct agents. In this case, one either can assign different labels to the same action if executed by different agents (and accordingly use these labels in the plan library), or introduce an additional sensor which "measures" the agent that is currently acting. If subtask are (arbitrarily) assigned to agents and possibly executed independently from each other, then one can use approaches which run individual plan recognizer for each agent and later fuse the results at a higher, more abstract level, like proposed by Hecking [Hec93].

Parallel, shared, and interleaved plans occur if more than one plan is executed by the observed agent or group of agents at the same time (see section 3.4). The recognition of more than one active plan at the same time leads to a combinatorial explosion of candidate hypotheses, as it is generally impossible to predict in advance whether a new observation belongs to an already active plan, starts a new plan, or both (also see subsection 5.2.7). In order to keep the computational complexity of the plan recognition problem tractable, most plan recognition approaches limit the number of plans that are allowed to be active at the same time. Examples of approaches which allow for the recognition of parallel respectively interleaved plans are [CG93, PW00, Sch03].

The plan recognition approach presented in this thesis assumes that the observed agent's plan selection and execution process is Markovian in order to allow for the use of Partially-observable Markov Decision Processes (POMDPs) to compute the expected utility of observation information (see chapter 7). Markovian in this context means that the future evolution of the process does not depend on its history (cf. discussion of Markov chains in section 4.3). This *memorylessness* hinders the recognition of *general recursive plans*, as one would have to remember where to "pick up" plan execution after the execution of a recursive subplan has finished. However, to our advantage the property of memorylessness holds for the practically relevant subclass of *tail recursive plans*. A recursion is called tail recursion if the recursive call (in our case the repeated invocation) happens at the end of a recursion step. This type of recursion is often used to express repeated iterations of a subplan and can be represented by simple loops in the plan library graph (see section 6.2).

A second limit related to the requirement of memorylessness concerns the recognition of *parallel* and *interleaved plans*. For the recognition of such plans the plan recognition system has to remember the progress of the first plan while the second plan is executed. This problem can be partly avoided

if the second plan that is executed does not have to be recognized. In this case, we can consider the actions that are executed as part of the second plan as unpurposeful actions and thus as not belonging to the original plan. Subsection 8.4.2 shows how to represent such actions in the plan library. This approach does not allow for the recognition of the second/interleaved plan, but it allows the observed agent to execute parallel plans without "disturbing" the recognition of the original plan.

5.2.6 Application Domains

Plan recognition techniques have been applied to a broad range of application domains. Due to the comparatively easy observability of user behavior early applications mainly focused on plan recognition in *software systems*. An example is the *SINIX Consultant* (SC) [WHK88], an intelligent help system for the SINIX operation system. It uses plan recognition techniques to detect inefficient or wrong use of the command line interface and gives unsolicited advice about correct or more efficient plans. As knowledge base the system uses a complex description of the commands and actions that might be performed in the SINIX domain, which also provides the systems plan library. A second example is provided by Thies, who applies plan recognition to assist users of graphical user interfaces [Thi94]. Thies represents plan libraries as multitrees and uses a spreading activation algorithm to propagate observation evidence from low-level action nodes via mid-level goal nodes to high-level plan nodes. Logical constraints on nodes are used to restrict the set of valid plans depending on the temporal and physical context. Thies uses a heuristic that disambiguates between competing plan hypotheses based on the number of activated child nodes.

Work in the domain of *natural language scene description* aims to describe and explain the behavior of agents in commentator-style natural language presentations. An example is the automated generation of comments for soccer matches from video data by Herzog and Retz-Schmidt [HRS89]. Here plan recognition is used to understand and interpret the behavior of players on the field. Nagel applies a cognitive visual system to generate natural language descriptions from video recordings of road traffic scenes [Nag04]. Here the system has to discover the intentions of cars respectively their drivers, like approaching or driving away from locations. Nagel uses Fuzzy Metric Temporal Horn Logic [Sch96] to represent both schematic and instantiated conceptual knowledge about the depicted scene and its temporal development.

Another large area of application is the *military* domain. Here, plan recognition is used to understand the plans and intentions of enemy troops. Rao

and Murray propose to use plan recognition to assess the enemies' mental-state, i.e. the beliefs, desires, plans, and intentions in air-combat modeling [RM94]. Heinze et al. extend this idea by incorporating machine learning to provide spatio-temporal recognition of environmental events and relationships [HGP99]. A formal description of tactical plan recognition in military applications is provided by Mulder and Voorbraak [MV03].

Newer applications of plan recognition focus on *public security* threads like *terrorism*. Jarvis and his colleagues use plan recognition techniques to support human intelligence analysts in processing security alert sets by automatically identifying the hostile intent behind them [JLM04]. A more general area is the protection of *critical infrastructures*. Banks et al. use plan recognition to discriminate "normal" and "anomalous" behavior in order to anticipate and predict threats that may impact critical operations or cause harm to people and infrastructure on a military base [BJH+07].

In the domain of *Ambient Assisted Living* (AAL) plan recognition has been applied by Aloulou and his colleagues to provide cognitive assistance to elderly users during lunch time [AFPB09].

Another application domain for plan recognition techniques is *shopping assistance* (also see subsection 5.1.3). In [Sch04], the smart shopping assistant application is presented, which supports a user in an instrumented real-world store. The application monitors the user's interactions with RFID-tagged products and infers the user's intentions. Recognized plans include buying known products without information need, investigation of and information gathering regarding unknown products, and multi product comparison shopping. The results of the plan recognition system are used by the shopping assistant to adapt the information display according to the user's needs, and to provide up-selling and cross-selling advertisements.

An increasingly relevant application area for plan recognition techniques is the domain of real-time strategy computer games. Albrecht et al. try to infer the human players' goals and plans in a *multi-user dungeon adventure game* with thousands of possible actions and locations [AZN98]. Their approach uses dynamic Bayesian networks with several different network structures which model the underlying domain to varying extents. Fagan and Cunningham apply a case-based plan recognition approach [KC01, KC03] to the classic Space Invaders game [FC03]. The authors show that case-based plan recognition can produce good prediction accuracy in real-time when working on a fairly simple game representation. Lee et al. apply plan recognition to optimize the state space that is used by dynamic scripting approaches in a real time strategy game [LKO08]. Dynamic scripting [SPSKP06] is applied to generate rule-based game scripts by using reinforcement learning.

5.2.7 Computational Complexity

Geib studied the theoretical complexity of plan recognition depending on certain properties of the underlying plan library [Gei04]. Geib investigates the complexity of plan recognition on the basis of an analysis of the number of explanations that any complete plan recognition algorithm must consider to explain a given sequence of observed actions. For this purpose Geib analyzes in which cases new observations require the generation of new explanations or allow to discard inconsistent observations. Geib distinguishes three distinct cases:

- *No new root goals*: Some actions only appear in the middle or at the end of some plan. Such observations do not introduce new goals but might increase the set of possible explanations if an observations "fits" into more than one explanation (and we don't know yet in which). The resulting number of explanations is bounded above by the number of existing explanations multiplied by the number of possible "attachment positions". On the opposite side, such observations might reduce the number of candidate explanations as they might lead to the exclusion of hypotheses which become inconsistent with the new observation.

- *New root goals with single leaders*: Some actions might additionally appear as single initial actions (so-called single *leaders*) at the beginning of a plan. When observed, they require the generation of an additional explanation to account for the possible start of a new plan. In this case, the set of candidate explanations increases linearly with the number of root goals for which the newly observed action is a leader.

- *New root goals with multiple leaders*: Some actions might appear together with some other actions as leaders of a certain plan. This happens if there exist no order constraints on the actions at the beginning of a plan. In this case, one has to create a new candidate explanation for each possible order of each possible subset of observed actions. The number of additional explanations in this case is bounded above by $(mn)^n$, where m is the number of unordered leaders and n is the number of plans which share these leaders.

Geib concludes, that the major factor influencing the runtime of plan recognition algorithms is the presence of shared, unordered lead actions, which have the potential to exponentially increase the number of candidate explanations. All other cases cause, at worst, an increase in the number of explanations that is linear in the number of possible attachment positions.

Geib thus proposes to use plan recognition algorithms that avoid an explicit enumeration of the complete explanation space before seeing any observation on plan libraries that have large numbers of repeated actions or long shared unordered plan leaders.

5.3 Sensor Selection

In the following subsections we present related work in the area of *sensor selection*. We describe existing application-dependent as well as application-independent sensor selection approaches and discuss aspects of the computation complexity of sensor selection.

5.3.1 Sensor Selection in Robotic Systems

One of the first applications of sensor selection techniques was in the field of *robotics*. Hovland and McCarragher describe a sensor selection approach for the real-time control of a planar robotic assembly task in a discrete event control framework [HM97]. They introduce the term *dynamic sensor selection*, which is the process of seeking new information when the current available information is inadequate. A so-called *sensor selection controller* chooses between different position and force sensors of a robot with differing characteristics in order to maximize information gain while keeping the sensory processing time low. The proposed controller uses lookup tables, which are generated off-line by a *stochastic dynamic programming* algorithm [Ros83]. Hence, this approach does not require any heavy computations at runtime and therefore is well suited to time-critical real-time applications.

The concept of *active vision* – sometimes also called *active computer vision* – is closely related to the area of sensor selection in the robotics domain. Here, the sensing system can actively influence the configuration of its sensors to adapt them to its current information needs. The system might for instance manipulate the viewport of its camera(s) in order to get better information from it/them. Marchand and Chaumette propose an active vision approach for the *3D reconstruction of static scenes* [MC99]. They propose two algorithms to control the automatic generation of camera motions: The first algorithm uses decision theory and Bayesian networks to implement an incremental reconstruction approach based on the use of a prediction/verification scheme. It allows the visual system to get a rather abstract, high level description of the observed part of the scene. Based on this information, the second algorithm then computes further viewpoints for the camera in order to ensure the complete reconstruction of the scene.

The concept of *active sensing* extends the idea of active vision to other types of sensors and system actions. The robot might for instance be able to execute special "probe" actions in order to test certain properties of its environment. A survey of the major methods for active sensing in robotics is given by Mihaylova et al. [MLB+02]. The authors discuss pros and cons of various model-based methods and pay special attention to different criteria for decision making. Denzler and Brown propose an information-theoretic sensor data selection approach for active object recognition and state estimation [DB02]. The authors use Shannon's *information theory* [Sha48] to select the sensor parameters that maximize mutual information based on the notion of "entropy", thus optimizing the information that the captured information conveys about the true state of the environment.

5.3.2 Sensor Selection in Wireless Sensor Networks

Another area of application for sensor selection techniques are *wireless sensor networks* (see subsection 2.3.1). Here, multiple battery-driven low-power low-cost devices equipped with sensors and a radio module are distributed throughout an environment. Through cooperation such networks can provide decentralized sensing and support applications.

An example for such an application is target tracking. The problem of *target tracking* is to minimize the error in estimating the position of a moving target. Wang et al. use an entropy-based sensor selection heuristic that chooses sensors based on the a-priori probability distribution of the target locations and the locations and sensing models of the candidate sensors [WYPE04]. Geometric properties of the environment and the involved sensors (like a camera's angle of view, coverage, etc.) are used by Isler and Bajcsy [IB05]. They consider a generic sensor model where all measurements can be interpreted as polygonal, convex subsets of the observed plane. Individual measurements are merged by intersecting the corresponding subsets, while the measurement's uncertainty corresponds to the size of the area of the intersection. Isler and Bajcsy propose an approximation algorithm in order to select sensors which minimize the resulting uncertainty.

A variant of the sensor selection problem in wireless sensor networks which host multiple different applications is the problem of *sensor-mission assignment*. Sensor-mission assignment involves the allocation of sensor and other information-providing resources to applications (missions) in order to cover the information needs of the individual tasks in each mission. In [GPJ+08], Gomez et al. approach this problem from a Semantic Web perspective. The core of their approach is a set of ontologies describing mission tasks and sensors. Semantic reasoning is used to recommend certain types of sensors that

are known to be suitable to solve the given sub-problems. These recommendations are used to constrain a search for available instances of sensors that can be allocated at mission execution-time to the relevant tasks.

A general survey of sensor selection methods in wireless sensor networks is provided by Rowaihy et al. [REJ+07]. The survey covers different classes of selection schemes: Coverage schemes, target tracking and localization schemes, single mission assignment schemes, and multiple mission assignment schemes. The authors also look at solutions to related problems from other areas and consider their applicability to sensor selection in wireless sensor networks.

5.3.3 Sensor Selection in Plan and Activity Recognition

The S-SEER system by Oliver and Horvitz [OH04] is an example of an activity recognition system that uses techniques for sensor selection to dynamically choose the set of evidence sources that is queried and analyzed with the goal of reducing the CPU time that is consume by the recognition system. The approach by Oliver and Horvitz is based on a decision-theoretic expected-value-of-information (EVI) measure which is based on a utility function $U(M_i, M_j)$, which returns the value of recognizing executed activity M_i as activity M_j (the value of U is maximal if $M_i = M_j$). An evidence source is activated if its EVI exceeds its costs (the net expected value of the evidence source is positive). Utility function U provides the main guideline for the sensor selection process, as it expresses the agent's preference for particular recognition results. In their experiments, Oliver and Horvitz use a very simple utility function that is based on an identity matrix, such that $U(M_i, M_j) = 1$ iff $M_i = M_j$, and 0 otherwise. As an alternative utility function the authors propose to define U based on the costs of misdiagnosis, where $U(M_i, M_j)$ equals the amount of dollars that the user would be willing to pay in order to avoid a misdiagnosis of M_i as M_j. However they do not explain how to derive these values or at least how to support the user in manually assessing the huge number of individual costs values that are required to completely define utility function U.

In the context of plan recognition applications, the problem of sensor selection has found virtually no attention by other researchers so far. One article which takes a first step into this direction is by Carberry and Elzer [CE07]. In this article the authors consider a system which tries to infer a graphic designer's intended message based on evidence in the form of communicative signals provided by graphics. The authors propose to perform evidence

analysis to better understand how different available evidence sources impact the recognition system's success. They present such an evidence analysis for communicative signals in the considered application scenario.

While this work illustrates that the need to carefully select information sources used for plan recognition has been identified by other researchers, to the best of our knowledge no research has been conducted on a general domain-independent utility model for observation information (and thus sensor selection) in plan recognition applications so far.

5.3.4 Generalized Sensor Selection

In contrast to application-specific sensor selection which exploits individual characteristics of particular application domains, generalized sensor selection approaches solve abstract variants of the sensor selection problem. Two classes can be roughly distinguished: (1) Approaches which try to maximize the value of a given, application-specific target functions which judges the "quality" of the selected set of sensors, and (2) approaches which try to minimize the estimation error given by some application-specific error measure.

In [BKG06] Bian et al. propose a generic sensor selection approach which utilizes a utility-theoretic target function to select a subset of promising sensors. The authors study the characteristics of two special classes of utility functions: submodular and supermodular functions. They propose an optimal solution algorithm for submodular functions and a linear-programming-based approximation approach for an important subclass of supermodular functions. Other generalized approaches have been considered in the area of dynamic systems [GCHM06, KP98, Osh94], sensor network management [REJ+07], hypothesis testing [DLT02], and discrete-event systems [JKG03].

In order to solve the problem of sensor selection in plan recognition applications we have chosen to follow the idea of generalized sensor selection by developing a utility model for sensor information in general plan recognition applications. This model then can be used in conjunction with generic utility-based sensor selection algorithms that can solve problems of the form presented in section 2.4. This design decision is motivated by the superior flexibility of this approach:

- **Free choice of sensor selection algorithm:** The separation of utility model and sensor selection logic allows to apply a large variety of generic sensor selection algorithms. This includes optimal, approximation, and anytime algorithms, which allows the designer of an environment to find a suitable trade-off between runtime and the "quality" of the found sensor sets. In addition, the separation of utility model and

sensor selection logic allows to benefit from future scientific advances in the area of utility-based sensor selection at little to no cost.

- **Reuse of utility model:** A general model of observation utility can be easily reused in other contexts. An example from another application area are *multi-agent auctions*. Here, multiple autonomous software agents compete for material or immaterial goods, services, or contracts. In order to know how much to bid in such an auction, an agent needs some way to know or estimate the utility of the offered item (for an overview on multi-agent auction strategies see [Gre03]). In the case where the subjects of an auction are information items, the proposed model can be used to control the bidding behavior.

For the implementation of our utility model we derive the *expected utility of observation information* by combining probabilistic plan recognition knowledge with information from a basic cost-reward model that "grounds" the computed utility values (see chapter 7).

5.3.5 Computational Complexity

The complexity of sensor selection depends on the number of candidate subsets that have to be considered. A naïve approach would be to evaluate the utility for all possible subsets by direct enumeration. Given a set of n sensors, there exist 2^n subsets. Clearly, direct enumeration is not practical unless n is very small. In its general form the problem of sensor selection is related to the knapsack-problem [KPP05], which is NP-hard to solve.

By cutting down the search space with techniques like *branch and bound* the sensor selection problem can be exactly solved without evaluating all possible subsets in the average case [Wel82, LW66]. However, in the worst case, one still has to consider all possible subsets. Therefore, several heuristics have been proposed to approximately solve the sensor selection problem. These include application specific heuristics as well as generic methods like evolutionary algorithms [YSK93] or randomized rounding [MR95].

In [BKG06] Bian et al. study the complexity of utility-based sensor selection for different classes of utility functions. They show that an optimal subset of sensors can be found in polynomial time in the case of submodular function. They prove that in the case of supermodular functions an optimal solution is NP-hard to find. For a practically important subclass of supermodular functions they give a linear-programming-based approximation algorithm which achieves an $O(\log n)$ approximation ratio.

5.4 Decision Models

In this section we discuss related work in the area of decision models. A special focus is put on how to determine the impact (and thus the utility) of individual input factors on the resulting decision-making. The general idea then is to formulate a decision problem that is derived from the general plan recognition problem, formulate it in a suitable decision model with observations as input factors, and finally derive the utility of individual input factors/observations from this model. This utility measure then can be used with generic sensor selection algorithms to solve the problem of sensor selection in plan recognition applications.

5.4.1 Expected Utility Networks

In [MS99], La Mura and Shoham introduce the concept of *expected utility networks*. Expected utility networks provide a new class of graphical representations in which – in contrast to belief networks – not only probabilities, but also utilities enjoy a modular representation. La Mura and Shoham introduce the notion of *conditional utility independence* to represent utilities in such networks. Similar to probabilistic inference, which involves the computation of conditional probabilities, expected utility networks allow for the computation of conditional expected utilities through so-called *strategic inference* – the reasoning process which underlies rational decision making.

The authors describe an example in which expected utility networks are used to model an agent's utilities in an auction. In the resulting network, La Mura and Shoham perform strategic inference in order to derive the optimal price to bid in the auction.

In contrast to other proposals which rely on additive notions of utility independence like [Sho97] or [BG95], La Mura and Shoham introduce a multiplicative notion. The authors argue, that the latter one is more intuitive because it is a close analogue of its probabilistic counterpart. Their notion is based on a *ceteris paribus* comparison operation for probabilities and utilities. Such an operation expresses how the probability respectively utility changes if an instantiation of a given subset of random variables is shifted away from some reference point, while the complementary set is held fixed. In this notion, two variables are called independent regarding expected utility (u-independent) if the increment in utility relative to the reference point is the product of the increments along each variable.

La Mura and Shoham argue, that u-independence is an attractive notion for two reasons: Firstly, it is intuitively understandable and applicable for people, as it only involves relevance considerations and order-of-magnitude

comparisons between utilities. Secondly, the analogy to probabilistic independence, which has a multiplicative notion too, allows for the construction of the aforementioned expected utility networks and associated inference mechanisms, which are simultaneously modular in probabilities, utilities, and expected utilities.

Expected utility networks have the advantage of allowing for an integrated representation of probabilities and utilities in a single model. On the downside they only allow modeling one-time decision situations. We will see in subsection 7.1.3 that our utility model is defined in terms of a sequential decision problem, which means that the resulting utility depends on a series of decisions. This disqualifies expected utility networks, as they do not allow for the representation of such continuous processes.

5.4.2 Preference Elicitation

The problem of *preference elicitation* concerns the process of learning preference or utility information from an agent, typically by issuing a set of queries that ask for the assessment of certain utility values. The gathered information then might be used by intelligent decision tools and autonomous agents that make or recommend decisions on behalf of the agent. Applications of elicitation processes are manifold, ranging from low-stakes decision processes (e.g. the control of user interaction with a web site) to critical decision assessment systems (e.g. clinical decision making).

The elicitation of preferences and utility functions is not trivial: Human users often cannot numerically asses these values, as humans generally only have an intuitive, qualitative understanding of utilities and preferences [Fre86, Hau00]. Secondly, not all information might be relevant to a certain decision problem. If some outcomes are impossible to show up, the utilities for those outcomes are useless to know. At last, the impact of some utility information on decision quality might be marginal, even if the information is relevant in general. If the cost of obtaining that information exceeds the benefit it provides, then this information can be safely ignored.

Chajewska, Koller, and Parr approach the problem of preference elicitation in [CKP00] by utilizing a prior probability distribution over the person's utility function, perhaps learned from a population of similar people. They judge the relevance of a concrete query for the current decision problem by its value of information. The authors propose an algorithm which interleaves utility elicitation and the analysis of the decision problem to allow both tasks to inform each other. At every step, the query with the highest utility of information is presented to the user. The process stops if the expected utility loss falls below a pre-specified threshold.

The approach of Chajewska et al. is criticized by Boutilier in [Bou02] for its myopic nature. A myopic approach can fail to ask the correct questions because it only decides for a single query at each time and thus neglects the value of future questions when determining the value of the current question. A myopic algorithm might underestimate the value of information in cases where value can only be obtained from a sequence of queries. Accordingly, Boutilier considers preference elicitation as a sequential decision problem by formulating it as a partially-observable Markov decision process (POMDP). In this POMDP, the state space is defined over the (infinite) set of all probability functions, while the action space is defined over the set of all possible queries. In order to deal with the continuous nature of the resulting state and action spaces Boutilier exploits the special structure of preference elicitation. He proposes an approximation approach to find the optimal value function.

The problem of preference elicitation is related to the problem of *model-based diagnosis*, in which a series of tests is performed in order to find an existing but hidden fault in a technical system [DP99]. Here, one of the problems is to decide on the best sequence of tests to diagnose the existing problem with minimal effort.

Both, the problem of preference elicitation and the problem of model-based diagnoses involve the need to decide on which query respectively test to trigger next in order to conclude the user's hidden preferences respectively the system's hidden fault. This relates to the problem of plan recognition where we have to infer the user's hidden plan and hence have to decide on which sensor (or set of sensors) to query next.

A second parallel is the formulation of observation source selection as a sequential decision problem as proposed by Boutilier. Similar to his approach we use POMDPs to model the utility of information sources. An important difference exists in the way we construct the POMDP. Instead of defining the POMDP state space over probability functions, we define it over states of the plan selection and execution process (see sections 6.2 and 7.2.3).

5.4.3 Decision Model Refinement

Decision models in general provide only an approximation of all the aspects that influence a complex decision process. In practical applications thus the question arises, whether it is reasonable to invest effort (and how much) into the refinement of a decision model or not. In [PH93], Poh and Horvitz investigate the value of extending the completeness of a decision model along three different dimensions of refinement: Quantitative refinement, conceptual refinement, and structural refinement.

Quantitative refinement aims at the refinement of the assessed numbers in a decision model. Poh and Horvitz describe two classes of quantitative refinement: Uncertainty refinement and preference refinement. *Uncertainty refinement* tries to increase the accuracy of probabilities, e.g. by tightening the bounds or second-order probabilities over probabilities in the considered decision model. *Preference refinement* means the refinement of numerical values representing the utilities associated with different outcomes of a decision.

Conceptual refinement is the refinement of the semantic content of one or more distinctions in a decision model. This kind of refinement seeks to modify the precision or detail with which certain aspects of the decision problem are represented in the model. For instance, in the umbrella example introduced in section 4.4, the decision maker might care about the actual kind of "rain", and want to distinguish between "drizzle", "light rain", and "rainstorm". Likewise, there might be more options than just whether to take an umbrella along or not. It might for instance be an alternative option to take a raincoat along or just to choose a water-repellent jacket.

Structural refinement is modeling effort that leads to the addition or deletion of conditioning variables or dependencies in a decision model. For example, a decision maker may discover that an expensive telephone-based weather service gives extremely accurate weather forecasts, and wishes to include the results of a query to this service in his decision analysis through the introduction of a new variable and additional dependencies.

Poh and Horvitz analyze all three kinds of model refinement and develop equations that describe the expected value (EVR) of continuing to refine a given decision model for each of the three dimensions. They further propose to use measures of EVR as guidance during the process of decision modeling in consultation settings, as well as within automated support systems that control the refinement of decision models.

Decision model refinement aims at increasing the "correctness" of a decision model along different dimensions. Here the concept of *expected value of refinement* is used as guidance for the refinement process. This appears to be similar to the problem of sensor selection in plan recognition applications, where sensor information helps us to refine our plan recognition model. Indeed we borrowed from Poh and Horvitz the idea to use expected utility as a decision-theoretic measure for information value. As we will see in section 7.1, our primary goal differs in that it is not to maximize the correctness of the plan hypothesis under all circumstances, but to maximize the added value that is generated based on the plan hypothesis.

5.4.4 Sensitivity Analysis

Sensitivity analysis investigates the relationship between input factors and outputs of a mathematical model. It studies how the variations in a model's output are related, qualitatively or quantitatively, to different sources of variation in the input of the same model. The statistical measures provided by sensitivity analysis are typically used to meet the following objectives:

- Identification of non-relevant input variables for the purpose of model simplification
- Identification of relevant input variables for subsequent calibration and optimization tasks, or prioritization of research
- Improvement of the understanding of the model structure: Interactions among variables, combinations of variables that result in high/low output values, etc.
- Assessment of model quality: Does output uncertainty depends on lack of knowledge in model parameters, on uncertainty in model structures, or on subjective assumptions?

Sensitivity analysis includes a variety of methods. Some of the most important classes of methods are: *Local methods* like the simple derivative of the output with respect to some input factor (taken at some fixed point in the input space) [Cac03], *sampling-based methods* which repeatedly execute the model for combinations of input values randomly selected from the (known) input distribution [HJSS06], *emulator-based methods* like Bayesian frameworks [Oak04] were the value of some output is treated as a stochastic process and estimated from the available computer-generated data points, methods based on *high dimensional model representations* (HDMR) which express outputs as linear combinations of terms of increasing dimensionality [Rab89], and *Monte-Carlo-based methods* which are used to identify regions in the input space that correspond to extreme values of the output [STCR04].

Areas of applications for sensitivity analysis are manifold and include physics and chemistry, financial applications, risk analysis, signal processing, neural networks, and any area where mathematical and/or scientific models are developed and applied.

While sensitivity analysis is well suited to discover and describe the dependencies between input and output parameters in a mathematical model, sensitivity analysis alone does not explain which output parameters are desirable to achieve (have a high utility). In addition, sensitivity analysis mostly focuses on the static structure of the model, while sensor selection should also consider the dynamic context of the current plan recognition situation.

5.5 Summary

Plan recognition approaches that rely on techniques for reasoning under uncertainty have manifold proved their suitability in practical plan recognition applications and – as discussed earlier – are the best way to deal with the uncertainty inherent to any plan recognition application. Consequently, the proposed state-based plan recognition approach that we present in the following relies on *probabilistic knowledge representation and reasoning* techniques, in particular *dynamic Bayesian Networks* (DBN) for modeling and inferencing the progress of the agent's plan selection and execution process, and *partially observable Markov decision processes* (POMDP) for reasoning about the value of observation information. The application of DBNs and POMDPs allows for a natural representation of state observations (see chapter 6) and a clear and consistent derivation of a utility model for observation information (see chapter 7).

As a drawback, the DBN and POMDP-based methods used in our approach do not natively allow for a *hierarchical representation* of the plan library, but instead use a flat state-based plan library. While approaches like proposed by Bui are similar to our approach and support hierarchical plans through the use of techniques like *Abstract Hidden Markov Memory Models* (AHMEMs), it is not obvious how to apply techniques like POMDPs (which we use to derive the utility of observation information) on the resulting AHMEM models. The lack of a hierarchical plan representation for the purpose of reasoning about abstract plans can be circumvented by introducing a meta model which maps states in a flat plan library to higher-order abstract plans. The probability of an abstract plan then is the sum of the probabilities of all mapped states. This process can be recursively repeated to model multi-level hierarchies. A second limitation concerns the recognition of parallel plans. Due to the memorylessness of the used probabilistic models only one plan at a time can be recognized by the proposed plan recognition approach. Although the parallel execution of a second plan cannot be recognized, its actions can be included as *unpurposeful actions* in the plan library (see subsection 8.4.2) such that the second plan's influence on the recognition of the original plan can be minimized.

An important aspect usually not considered in the context of existing plan recognition approaches is the role of the external system and its use of the plan recognition results to provide some kind of value added service to the agent and/or the owner of the system. In section 7.1 we discuss that it is important to consider the role of the external system in order to estimate the utility of observation information in plan recognition applications. Hence the applied plan recognition approach should account for possible influences

of the external system's behavior on the agent's plan selection and execution process, especially if this process is influenced by the assistance that is provided by the external support system. This demand is fulfilled by the proposed plan recognition approach by representing the external system's actions as a distinct node in our refined plan recognition DBN (see section 7.3). Existing plan recognition approaches usually do not consider the influence of external systems' actions in their reasoning. An exception again is Kautz' approach, which can account for provided support if the according dependencies are represented by according logic rules in the plan library rule set.

A central property which none of the plan recognition approaches that we are aware of possesses is the support of external applications in their decision making on the optimal behavior in response to a new plan recognition hypothesis. Classical plan recognition systems provide a (possibly rated) hypothesis of plan candidates, but do not further care about what happens with this information. Instead, plan recognition approaches with all their knowledge about the agent and its behavior could better support the external higher-order system in its utilization of the results, e.g. by predicting possible outcomes of candidate support actions given the current plan knowledge. We follow this idea by including a support model into our plan recognition tool set, which allows to provide the support system not only with plan hypotheses, but also with a proposal on the optimal support action to execute as response to the inferred plan information.

A second and even more central property which none of the plan recognition approaches that we are aware of possesses is the support or an eventually applied sensor selection strategy. The need for sensor selection in real-world plan recognition applications already has been motivated in section 1.1 and provides one of the main motivations for the research presented in this thesis. Plan recognition can (to a varying extend) predict the actions that an agent might execute in the future. Intuitively, this knowledge should be usable to point a sensor selection strategy to promising sensors, or more general by using plan knowledge to support a sensor selection process by providing an estimate of the expected usefulness of sensors.

Instead of developing an application-specific sensor selection strategy, the choice was made to approach the problem by developing a utility model for sensor information in general plan recognition applications. This model then can be used in conjunction with generic utility-based sensor selection algorithms that can solve problems of the form presented in section 2.4.

Table 5.1 summarizes the most relevant similarities and differences between the previously presented existing plan recognition approaches and our proposed resource-aware plan recognition toolkit RePReTo, that we present in the following chapters.

	Kautz 1986	Charniak & Goldman 1993	Bauer 1993	Thies 1994	Pynadath & Wellman 2000	Schneider 2003	Bui 2003	Schneider 2009
Applied Calculus[1]	PL	BN	DS	FOPL	PSDG/BN	PRM	AHMEM	DBN/POMDP
Probabilistic Reasoning	-	yes	yes	-	yes	yes	yes	yes
Hierarchical Plan Libraries	yes	yes	-	yes	yes	yes	yes	(yes)[2]
Recursive Plans	(yes)[3]	yes	-	-	yes	-	-	partly[4]
Interleaved Plans	-	yes	-	yes	yes	yes	-	partly[5]
Reasoning About State Observations	(yes)[3]	-	-	-	-	-	yes	yes
Considering Influence of Support	(yes)[3]	-	-	-	-	-	-	yes
Assisting Choice of Optimal Support	-	-	-	-	-	-	-	yes
Support of Sensor Selection	-	-	-	-	-	-	-	yes

1) PL: Propositional Logic; BN: Bayesian Networks; DS: Dempster-Shafer; FOPL: First-Order Predicate Logic; PSDG: Probabilistic State-Dependant Grammars; RM: Probabilistic Relational Models; AHMEM: Abstract Hidden Markov Memory Models; DBN: Dynamic Bayesian Networks; POMDP: Partially Observable Markov Decision Processes

2) Not natively, but via a meta model that links plan execution states to abstract higher-order plans

3) Not natively, but can be encoded in the plan library rule set

4) Tail-recursive plans can be expressed through loops in the plan library graph

5) Cannot be recognized, but ignored if modeled as unpurposeful actions (see subsection 8.4.2) in plan library

Table 5.1: Comparison of existing plan recognition approaches with the proposed resource-aware plan recognition toolkit (RePReTo)

.

Prediction is difficult, especially about the future.
Niels Bohr (1885–1962)

6 A State-Aware Plan Recognition Approach for Instrumented Environments

In this chapter we propose a novel state-aware plan recognition approach which addresses the problem of state observations in instrumented environments (see section 1.1). At first we motivate the general idea of the presented plan recognition approach (see section 6.1). Next we formally define the underlying plan selection and execution model that represents the agent's plan library (see section 6.2) and introduce the sensor model which links observations to abstract plan execution states (see section 6.3). Finally we show how to link both models to realize a state-aware plan recognition system (see section 6.4) and explain how to use this system to predict intended goals and anticipate future actions (see section 6.5).

6.1 Motivation and General Idea

Traditional plan recognition approaches usually perform *action-based plan recognition*, which tries to infer the executed plan based on a sequence of observed low-level agent actions. Such actions typically have a symbolic representation and provide a certain degree of abstraction from their concrete realization. In virtual environments like computer games, operating systems, or virtual reality, such symbolic actions are most often easy to observe through software probes, which have full and direct access to the system's internal processes. This in particular holds for event-based software systems where agent actions directly correspond to the occurrence of software events.

This situation changes fundamentally if plan recognition is applied in real-world environments, where the system has to rely on physical sensors to observe the agent's behavior. Due to the underlying principles of operation,

the majority of physical sensors sense (partial) state information instead of actions. Examples are sensors that measure temperature, humidity, intensity of light, noise, pressure, location, or proximity. At best they can indirectly infer the execution of particular actions from the observation of resulting state changes. Figure 6.1 illustrates the relationship between states and actions with a simple example: Consider the execution of a symbolic action "get flour from cupboard" in a kitchen environment[1]. While it is difficult to directly observe this action with a dedicated sensor, one might be able to detect the relocation of an RFID-tagged pack of flour via RFID readers/antennas (see subsection 2.3.2) in the cupboard and under the countertop. While in this example it is easy to conclude from the observed state change that a "get flour from cupboard" action was executed, such conclusions are not always possible or valid in the general case. This especially holds if more complex actions than the moving of objects should be observed, even more if this requires the fusion of information provided by multiple different sensors.

Figure 6.1: Example of a state change that is caused by the execution of an action. Depending on their principles of operation, physical sensors usually cannot detect the action itself, but only the resulting change in state.

The central idea of *state-based plan recognition* is to ease the problem of state observations by explicitly considering and representing state information in the plan recognition system's knowledge models and reasoning. The explicit representation of state information has two advantages: It relieves the infrastructure from the burden of inferring explicit symbolic agent actions from observed state changes, and it makes plan recognition more robust to

[1] Action "get flour from cupboard" is an abstraction of its concrete realization, which comprises the execution of a sequence of "reach", "grab", and "release" actions (and even these might be further subdivided into several "move muscle" actions, and so on).

ambiguous or missed observations of actions. The latter originates from the fact that the new (partial) state that results from the execution of an action usually can be observed for a longer time than the action itself.

The central concept of state-aware plan recognition is the so-called *plan execution state space* (denoted $\mathcal{V}$), which is a finite set of symbolic states. Every plan execution state space contains an obligatory *idle* state, which represents the case that the agent has not yet started with the execution of a plan. The plan execution state space furthermore contains a set of terminal states (denoted $\mathcal{G}$), where each terminal state represents the termination of a certain plan. Plans might be completed successfully, in which case the terminal state corresponds to the attainment of the goal that is associated with the executed plan. Plan execution might also fail, in which case the terminal state corresponds to the abortion or failure of the executed plan.

In order to move from the *idle* state towards some desired terminal state, the agent repeatedly chooses and executes actions from its finite *action repertoire* (denoted $\mathcal{A}$). Intermediate plan execution states represent the progress of the agent's plan selection and execution process. We assume, that each intermediate plan execution state relates to a characteristic subset of the world state, which exists as a result of the actions which have been executed so far. This state might include the agent's mental state, the state of artifacts or technical systems, and the physical state of the environment itself.

With such a state-based representation, the problem of plan recognition becomes the problem of estimating the agent's current (hidden) plan execution state given a sequence of sensor observations. The resulting *plan recognition hypothesis* then is a probability distribution over the plan execution state space that expresses the system's belief about the current state and progress of the plan selection and execution process. All other information like a prediction on possibly targeted goal states and planned future actions can be easily derived from the hypothesis and plan selection and execution model through probabilistic reasoning, as we show in section 6.5.

6.2 Plan Selection and Execution Model

6.2.1 Formal Definition

This section introduces the formal description of the agent's plan selection and execution model, which represents the agent's *plan library*. The model aims to provide a statistical description of the agent's behavior, and as such serves as the foundation for the plan recognition system's reasoning. As motivated in the previous section, the central idea of the proposed model

is to express the agent's plan selection and execution process as a sequence of state transitions in the plan execution state space. Each sequence starts in the *idle* state and ends in one of multiple possible terminal states. The observable state transitions result from the execution of a sequence of actions, which the agent chooses in order to reach an intended goal. A plan then is a path through the plan execution state space, which starts in the *idle* state and ends in a terminal state. The set of plans represented by a plan library then simply is the set of all existing paths of this structure.

The formal model that will be presented in the following assumes that the agent's process of plan selection is tightly interwoven with the process of plan execution. This assumption is justified by the non-deterministic nature of plan execution in real-world environments, which results in partially unpredictable results of executed actions. Reasons for this include:

- **Mishap:** Actions might lead to undesired results due to the agent's limited skills or bad luck.
- **Missing Knowledge:** Actions might have unexpected outcomes because some unknown precondition was not met.
- **General Nondeterminism:** The effects of actions are generally not fully predictable, independently of the agent's skills or knowledge.

For these and other reasons, the execution of the same action in the same state might lead to a transition to one of several possible successor states. Depending on the resulting state, the agent might then have to choose different follow-up actions to either continue as planned, to fix an eventually occurred problem to still reach the intended goal, to reach an alternative goal if reaching the original goal is not possible any longer, or to abort the plan execution process at all. To account for this dynamic replanning, we model each plan execution step as a two-phase process:

1. **Plan Selection Phase:** The agent chooses an action from its action repertoire that it assumes will bring it from the current state closer towards the desired goal state.
2. **Plan Execution Phase:** The agent executes the action selected in step 1, which then brings it to one of several possible successor states (not necessarily the intended one).

This process is repeated until a terminal state is reached. It is assumed, that the action which is chosen (and thus the set of possible resulting desti-

nation states) only depends on the current state[2]. In other words, we assume that the description of the present state fully captures all information that might influence the future evolution of the plan execution process. With this assumption, the plan selection and execution process is a Markov chain (see section 4.3).

Formally, the agent's plan selected and execution process can be represented as an extended *probabilistic automaton* [Rab63], which generalizes the concept of a Markov chain. A probabilistic automaton is an extension of a *non-deterministic finite automaton* and is defined as follows:

Definition 6.2.1. A **probabilistic automaton** is a tuple $(\mathcal{V}, \mathcal{G}, \mathcal{A}, t, v_0)$, where

- $\mathcal{V}$ is a finite set of states
- $\mathcal{G} \subset \mathcal{V}$ is the set of accepting (or terminal) states
- $\mathcal{A}$ is a finite set of input symbols
- $t : \mathcal{V} \setminus \mathcal{G} \times \mathcal{A} \times \mathcal{V} \mapsto [0, 1]$ is the transition function, where $t(v, \alpha, v')$ returns the probability that an input symbol α which is observed in state v results in a transition to state v'
- $\forall_{v \in \mathcal{V} \setminus \mathcal{G}, \alpha \in \mathcal{A}} : \sum_{v' \in \mathcal{V}} t(v, \alpha, v') = 1$
- $v_0 : \mathcal{V} \mapsto [0, 1]$ is a probability mass function which returns the probability of the automaton being in a given initial state

If we understand the set $\mathcal{V}$ of automaton states as the plan execution state space and the set $\mathcal{A}$ of automaton input symbols as the agent's action repertoire, then a probabilistic automaton can describe the state transitions that occur as the result of the occurrence of certain input symbols (respectively the execution of certain actions). However, this description still lacks a representation of the action selection process, which is to describe which actions are executed by the agent in which states with which probability. For this purpose, the formal definition of the plan selection and execution model (denoted in the following as a plan library L) extends the probabilistic

[2]In fact, the chosen action also depends on the intended goal, but as we cannot directly observe this aspect, we assume that this dependency is implicitly represented by the action/transition probabilities associated with each state.

automaton model with an additional action selection function s. Formally, a plan library is defined as:

Definition 6.2.2. A **plan library** is a tuple $(\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$, where

- $\mathcal{V}$ is a finite set of plan execution states
- $idle \in \mathcal{V}$
- $\mathcal{G} \subseteq \mathcal{V} \setminus \{idle\}$ is the set of terminal states
- $\mathcal{A}$ is the agent's action repertoire
- $s : \mathcal{V} \setminus \mathcal{G} \times \mathcal{A} \mapsto [0, 1]$ is the action selection function, where $s(v, \alpha)$ returns the probability that the agent executes action α if the current state is v
- $\forall_{v \in \mathcal{V} \setminus \mathcal{G}} : \sum_{\alpha \in \mathcal{A}} s(v, \alpha) = 1$
- $t : \mathcal{V} \setminus \mathcal{G} \times \mathcal{A} \times \mathcal{V} \mapsto [0, 1]$ is the transition function, where $t(v, \alpha, v')$ returns the probability that an execution of agent action α in state v results in a transition to state v'
- $(\mathcal{V}, \mathcal{G}, \mathcal{A}, t, v_0)$ is a probabilistic automaton with $v_0(idle) = 1$ and $v_0(v \neq idle) = 0$ otherwise

The dependencies of actions and state transitions that are defined in a plan library can be visualized as a directed graph, where the set of nodes corresponds to the plan execution state space, and labeled edges correspond to transitions that result from the execution of single agent actions. The label of an edge denotes the executed action and the combined probability of executing the associated action and resulting in the target state. Note, that this implies that the sum of probabilities of all edges leaving any non-terminal node equals to one. The resulting graph is called a *plan library graph* (see next section for a concrete example of a plan library graph).

6.2.2 Example Plan Library

In this section we present a simplified example plan library from the domain of cooking. This plan library provides the foundation for further examples presented in the remainder of this thesis. We start by presenting the plan

library graph and describing the represented plans before we give the formal description of the plan library according to the above definitions.

Figure 6.2 shows the plan library graph for our example plan library which represents four general groups of plans (marked with dashed boxes) with multiple variants each:

- The preparation of *fried eggs* with two possible goal states *have good fried eggs* and *have bad fried eggs*
- The preparation of *scambled eggs* with only a single goal state *have scrambled eggs*
- The preparation of *mayonnaise* with four possible goal states *have inedible mayonnaise* (e.g. caused by a contamination with salmonella due to the use of expired eggs), *dispose eggs* (as an abortion of the original plan), *have tasty mayonnaise* (a successfully completed plan), and *have untasty mayonnaise* (e.g. caused by improper mixing of the ingredients)
- Visiting and leaving the kitchen for some other reason with two possible goal states *out of kitchen with lights off* and *out of kitchen with lights on*, where the latter goal state represents the case that the user has forgotten to turn off the lights

All plans start with the user beeing outside of the kitchen (represented by state *idle*). At some point in time the user enters the kitchen, possibly turns on the lights if it is dark in the kitchen (represented by state v_1), performs some other activities, and eventually moves to the fridge. Now, the user decides whether to start the preparation of food by picking eggs from the fridge, or instead to leave the kitchen (possibly after taking something else from the fridge). Before leaving the kitchen, the user possibly turns off the lights. In case the user decides to prepare fried or scrambled eggs, she picks a pan and starts with the preparation. In case she decides to prepare mayonnaise, she picks a whisk and starts mixing the required ingredients[3].

During the execution of the intended plan different problems may occur. The resulting erroneous plans are represented as plan alternatives in the plan library.

[3]For reasons of clarity we have omitted in our example the actions that are required to collect the remaining tools and ingredients.

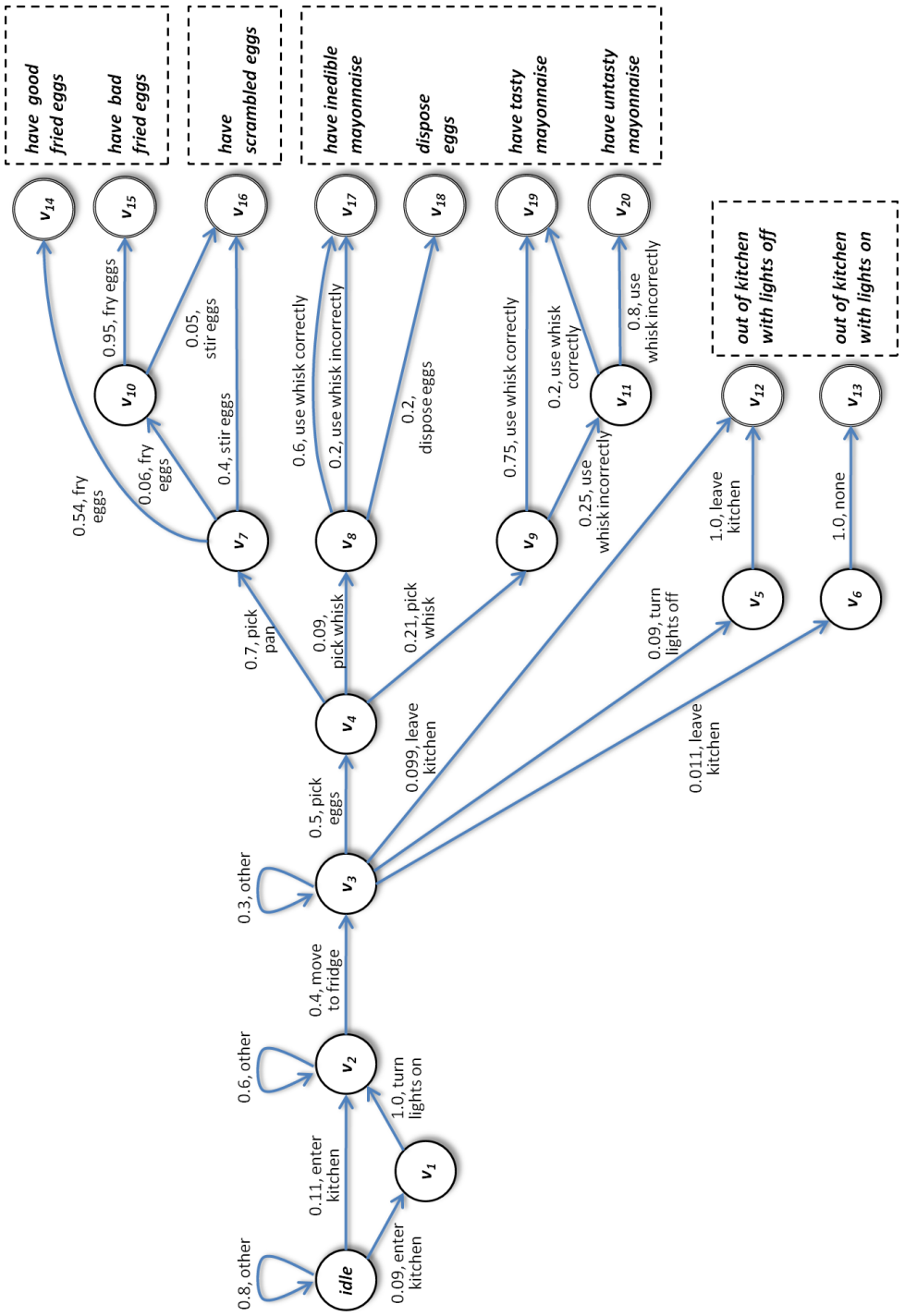


Figure 6.2: Example plan library graph from the cooking domain.

For instance while frying eggs in the pan the eggs might accidentally get scrambled (represented by state v_{10}). In this case the user might choose to revert her original plan and instead change to the alternative plan of preparing scrambled eggs. Another example of an erroneous plan is preparing mayonnaise with expired eggs (represented by state v_8 and following). If the user notices this problem, she might abort the plan and dispose the eggs. Otherwise, she might end up with inedible mayonnaise due to salmonella contamination. Furthermore the user might incorrectly use the whisk to mix ingredients (state v_{11}). If the user does not correct this error (represented by transition $v_{11} \rightarrow v_{19}$), she ends up with untasty mayonnaise.

Recall that edges in the plan library graph correspond to state transitions that occur due to the execution of user actions. Each edge in the graph is labeled with the executed action and a probability which is the product of the action selection probability and the action-dependent transition probability. For our example plan library, the probabilities that particular actions are selected in a certain state are given by the values of function s in Table 6.1. The probabilities of resulting transitions depending on the current state and chosen action are given by the values of function t in Table 6.2.

Each path from the *idle* state to a goal state represents a plan in the plan library. Without considering the loops introduced by the "*other*" actions in states $idle, v_2$, and v_3 the presented example plan library describes a total of 26 plans. For instance four plans exist to reach the goal *have scrambled eggs*:

1. [enter kitchen, move to fridge, pick eggs, pick pan, stir eggs]
2. [enter kitchen, move to fridge, pick eggs, pick pan, fry eggs, stir eggs]
3. [enter kitchen, turn lights on, move to fridge, pick eggs, pick pan, stir eggs]
4. [enter kitchen, turn lights on, move to fridge, pick eggs, pick pan, fry eggs, stir eggs]

With additionally considering the loops introduced by the "*other*" actions the plan library describes indefinitely many plans, which differ from each other by the number of "*other*" actions that are executed in states $idle, v_2$, and v_3.

The a-priori probability of a plan can be computed by multiplying the individual probabilities associated with each involved transition. For the first plan for scrambled eggs mentioned above the probability (with considering the *other* actions in states $idle, v_2$, and v_3) for instance is

$$0.11 \cdot 0.4 \cdot 0.5 \cdot 0.7 \cdot 0.4 = 0.00616$$

According to Definition 6.2.2 from the previous subsection, our example plan library can be formally described by the tuple $(\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ with

State v	**Agent Action** α	$s(v, \alpha)$
idle	enter kitchen	0.20
	other	0.80
v_1	turn lights on	1.00
v_2	move to fridge	0.40
	other	0.60
v_3	pick eggs	0.50
	leave kitchen	0.11
	turn lights off	0.09
	other	0.30
v_4	pick pan	0.70
	pick whisk	0.30
v_5	leave kitchen	1.00
v_6	idle	1.00
v_7	fry eggs	0.60
	stir eggs	0.40
v_8	use whisk correctly	0.60
	use whisk incorrectly	0.20
	dispose eggs	0.20
v_9	use whisk correctly	0.75
	use whisk incorrectly	0.25
	dispose eggs	0.00
v_{10}	fry eggs	0.95
	stir eggs	0.05
v_{11}	use whisk correctly	0.20
	use whisk incorrectly	0.80
otherwise		0.00

Table 6.1: Probability values of example action selection function s.

Start State v	**Action** α	**End State** v'	$t(v, \alpha, v')$
idle	enter kitchen	v_1	0.45
		v_2	0.55
	other	*idle*	1.00
v_1	turn lights on	v_2	1.00
v_2	move to fridge	v_3	1.00
	other	v_2	1.00
v_3	pick eggs	v_4	1.00
	leave kitchen	v_6	0.10
		v_{12}	0.90
	turn lights off	v_5	1.00
	other	v_3	1.00
v_4	pick pan	v_7	1.00
	pick whisk	v_8	0.10
		v_9	0.90
v_5	leave kitchen	v_{12}	1.00
v_6	idle	v_{13}	1.00
v_7	fry eggs	v_{10}	0.10
		v_{14}	0.90
	stir eggs	v_{16}	1.00
v_8	use whisk correctly	v_{17}	1.00
	use whisk incorrectly	v_{17}	1.00
	dispose eggs	v_{18}	1.00
v_9	use whisk correctly	v_{19}	1.00
	use whisk incorrectly	v_{11}	1.00
	dispose eggs	v_{18}	1.00
v_{10}	fry eggs	v_{15}	1.00
	stir eggs	v_{16}	1.00
v_{11}	use whisk correctly	v_{19}	1.00
	use whisk incorrectly	v_{20}	1.00
otherwise			0.00

Table 6.2: Probability values of example transition function t.

- $\mathcal{V} = \{idle, v_1, \ldots, v_{20}\}$
- $\mathcal{G} = \{v_{12}, \ldots, v_{20}\}$
- $\mathcal{A} = \{$enter kitchen, leave kitchen, turn lights on, turn lights off, move to fridge, pick eggs, pick pan, fry eggs, stir eggs, pick whisk, use whisk correctly, use whisk incorrectly, dispose eggs, other, idle$\}$
- The values of action selection function s are given in Table 6.1
- The values of transition function t are given in Table 6.2

We return to the presented cooking example in later sections when we introduce the sensor model or our utility model for observation information.

6.2.3 DBN Formulation of Plan Execution Process

Above we mentioned that the plan selection and execution process described by a plan library $(\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ is a Markov chain, and as such can be represented by a DBN (see section 4.3). In our case this DBN has the structure shown in Figure 6.3. Each time slice corresponds to one iteration of the plan selection and execution process. The domain of the *Old State* and *New State* variables equals the plan execution state space $\mathcal{V}$. The two variables represent the plan execution state before and after the execution of a single action. The domain of the *Agent Action* variable equals the agent's action repertoire $\mathcal{A}$, and represents the action that is chosen and executed by the agent in the considered plan execution step. The conditional probability tables for the involved variables are derived from the plan library's functions s and t. The roll-up function from *New State* to *Old State'* is the identity function.

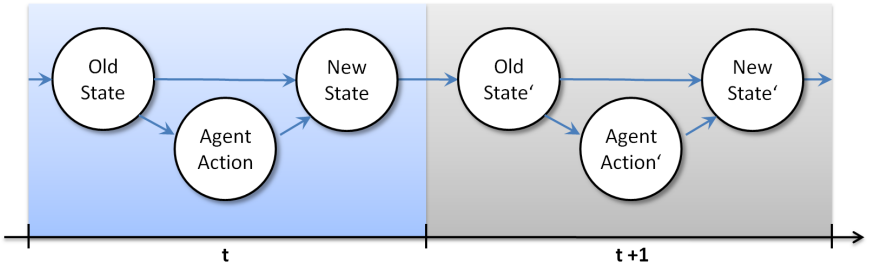


Figure 6.3: Evolution of the agent's plan selection and execution process represented as a dynamic Bayesian network with three nodes.

Functions s and t do not explain what happens once the agent's plan execution process reaches a terminal state. For now it should be assumed, that plan execution ends in this case, and that in following time slices the agent

always executes a pseudo action *null*, which neither results in any observations nor causes any state changes. Formally, this behavior is represented by the following modified action selection and transition functions s' and t':

$$s'(v,\alpha) = \begin{cases} 1 & \text{if } (v \in \mathcal{G}) \wedge (\alpha = null) \\ 0 & \text{if } (v \in \mathcal{G}) \wedge (\alpha \neq null) \\ s(v,\alpha) & \text{otherwise} \end{cases}$$

$$t'(v,\alpha,v') = \begin{cases} 1 & \text{if } (v \in \mathcal{G}) \wedge (\alpha = null) \wedge (v = v') \\ 0 & \text{if } (v \in \mathcal{G}) \wedge ((\alpha \neq null) \vee (v \neq v')) \\ t(v,\alpha,v') & \text{otherwise} \end{cases}$$

Additionally, we have to change the domain of the *Agent Action* variable to the agent's extended action repertoire $\mathcal{A} \cup \{null\}$. This extension now allows to define the conditional probability tables of the DBN as: $P_{\text{agent action}}(\alpha|v) = s'(v,\alpha)$, $P_{\text{New State}}(v'|\alpha, state) = t'(v,\alpha,v')$, and $P_{\text{Old State}}$ is the identity matrix which simply copies the distribution of the *New State* variable to the *Old State* variable in the next time slice.

Given a state-based representation of the plan selection and execution process, a plan recognition hypothesis can be understood as a probability distribution over the library's plan execution state space $\mathcal{V}$. In the DBN representation that is introduced above, this equals to the probability distribution over the values of the *New State* node. Formally, a state-based plan recognition hypothesis then is a probability mass function:

Definition 6.2.3. A **state-based plan recognition hypothesis** h_L regarding some plan library $L = (\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ is a probability mass function $h_L : \mathcal{V} \mapsto [0,1]$, where $h_L(v)$ returns the probability that the agent's plan selection and execution process is currently in state v. The infinite **set of all hypotheses** regarding some plan library L is denoted $\mathcal{H}_L$.

6.3 Sensor Model

6.3.1 Formal Definition

The plan selection and execution model that we introduced in the previous section describes the agent's behavior on an abstract level. In order to estimate the current plan execution state from real-world sensor observations, we need to know the (partial) dependencies between symbolic plan execution states, executed agent actions, and resulting sensor observations. The purpose of the *sensor model* is to provide a statistical description of these dependencies. A probabilistic approach is chosen, because sensor readings often non-deterministically depend on the executed actions and resulting states of the environment. Reasons for this include:

- **Inherent Inaccuracy**: Depending on their principle of construction the accuracy of different types of sensors may differ. It is generally hard and often even impossible to construct sensors which always work absolutely error-free under all circumstances.
- **Individual Inaccuracy**: Even exact sensors might return erroneous and unpredictable data for a variety of reasons, including improper calibration, zero drift, or general technical damage and malfunction.
- **Incomplete Knowledge**: Even if a sensor works perfectly, there still might be factors which influence the resulting readings which we are either unaware of, or which we cannot or want not consider in our sensor model.

In the sensor model that is presented in the following, the stochastic dependencies between executed agent actions, plan execution states, and sensor observations are described by a set of conditional probabilities. These probabilities reflect and abstract from the above mentioned uncertainty about the true value of the measured variables.

The presented sensor model assumes that depending on each individual sensor's operation principle and implementation the resulting sensor readings are influenced either by the executed action, the resulting plan execution state, the concrete resulting transition, or any combination of these. Considering all of these cases allows representing sensors which sense state information as well as sensors which sense the execution of actions. Definition 6.3.1 formally introduces sensor models.

Sensor models that are described in the form defined above can be expressed as Bayesian networks (see section 4.3) with the general structure

Definition 6.3.1. A **sensor model** is a quadruple $(\mathcal{S}, \mathcal{V}, \mathcal{A}, P)$, where

- $\mathcal{S} = \{s_1, \ldots, s_n\}$ is a finite set of sensors
- $dom(s_i)$ is the domain of $s_i \in \mathcal{S}$ and corresponds to the (discrete or continuous) set of readings that s_i can return
- $\mathcal{V}$ is the set of plan execution states that is covered by the sensor model
- $\mathcal{A}$ is the set of agent actions that is covered by the sensor model
- $P = \{p_1, \ldots, p_n\}$ is a finite set of probability functions, where $p_i : \mathcal{V} \times \mathcal{A} \times \mathcal{V} \times dom(s_i) \mapsto [0, 1]$ and $p_i(v, \alpha, v', r)$ returns the probability that sensor s_i generates reading r if a transition from old state v to new state v' occurs as a result of executing action α
- $\forall_{i \in \{1,\ldots,n\}, v \in \mathcal{V}, \alpha \in \mathcal{A}, v' \in \mathcal{V}} : \sum_{r \in dom(s_i)} p_i(v, \alpha, v', r) = 1$

shown in Figure 6.4, left. In these networks, each sensor is represented by a distinct random variable. The domain of a sensor variable equals the according sensor's set of possible readings, and depending on the type of sensor is either discrete or continuous. For reasons of simplicity we assume in the following that sensors only return a finite (thus discrete) set of possible readings. Three additional random variables represent the action that is executed by the agent (where the domain of the *agent action* variable equals $\mathcal{A}$), the old state of the execution process before the action was executed (precondition), and the new states of the environment after the action was executed (postcondition). The domains of the *Old State* and *New State* variables equal $\mathcal{V}$). The conditional probability table of each sensor variable $Sensor_i$ can be directly inferred from the sensor model's reading probability function p_i.

The given Bayesian network representation of the sensor model is only valid if we assume that all sensors are conditionally independent of each other. This means that every sensor's reading only depends on the executed action and the resulting state transitions, but not on the readings eventually returned by other sensors. If for any reason a set of sensors is conditionally dependent on each other, then this set has to be represented in the model as a single abstract *super sensor*, whose domain and conditional probability table is chosen in a way that accounts for the conditional dependencies of the contained basic sensors. Figure 6.4 (right) shows an example of such a super sensor i, which represents m conditionally dependent basic sensors.

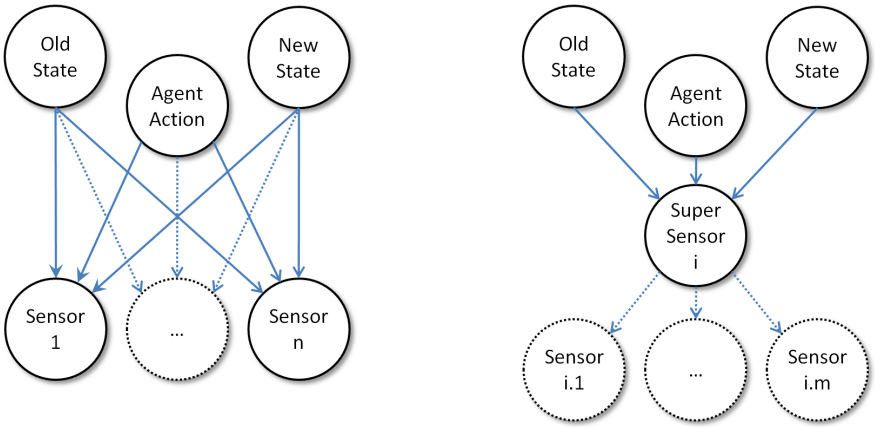


Figure 6.4: A Bayesian network visualization of the generic sensor model. The model assumes that in general sensors are conditionally independent (left). A set of conditionally dependent sensors can be modeled as a single virtual "super sensor", which abstracts from the individual sensors' dependencies (right).

In the graphical representation of the sensor model, super sensors can be easily introduction by adding according nodes and edges to the Bayesian network as shown on the right in Figure 6.4. In the formal representation of the sensor model which was introduced in Definition 6.3.1, super sensors can either be introduced by extending the formal definition, or by collapsing the super sensor subnetwork to a single random variable and conditional probability table, and to further treat this collapsed sensor network as a single "normal" sensor. In the rest of this thesis, we assume that the latter option is taken, and thus the sensor model always has the structure shown on the left of Figure 6.4.

6.3.2 Example Sensor Model

In the following we extend our example plan library from subsection 6.2.2 by a sample sensor model. Our sensor model assumes that the following six sensors exist in the environment:

- A *steerable camera* with different computer vision algorithms which can be operated in three configurations: (a) looking at the kitchen's door the camera can detect whether the user enters or leaves the kitchen, (b) looking at the fridge it can detect if someone is standing in front of the fridge, and (c) looking at the stove the camera can detect if fried

Sensor s	**Domain** $dom(s)$
cam door	{entered room, left room, no movement}
cam fridge	{in front of fridge, not in front}
cam stove	{fried egg ok, fried egg not ok, no fried egg}
wsn light	{natural light, electric light, darkness}
wsn whisk	{correct use, incorrect use, no use}
dpm eggs	{eggs ok, eggs expired, no information}
rfid fridge	{eggs not in fridge, eggs in fridge}
rfid counter	{pan present, whisk present, neither present}

Table 6.3: Domains of sensors in the example sensor model.

eggs in the pan look good or scrambled. According to the discussion in subsection 2.4.2 the steerable camera is an example of a configurable sensor and is represented in the presented sample sensor model by three individual sensors "*cam door*", "*cam fridge*", and "*cam stove*".

- Two *wireless sensor network nodes*, the first denoted "*wsn light*" is deployed in the kitchen and allows to detect the light level in the room via a photo diode. From the light frequency the sensor additionally can conclude whether present light is natural or electric. The second node denoted "*wsn whisk*" is attached to the whisk utensil and uses accelerometers to control the correct application of the whisk while mixing ingredients.

- Product-related information might be available via a *digital product memory* (DPM). DPMs record information about an object's properties and history on a per-instance level (cf. [Sch07b]). Hence DPMs can be considered as sensors which report about the state of an object. In our example the sensor "*dpm eggs*" represents a query to the digital product memory of the currently handled eggs that retrieves whether the eggs' "best before" date expired.

- Two *RFID readers*, one in the fridge to detect which ingredient is removed (denoted "*rfid fridge*") and one under the counter top to detect which objects are place on the counter (denoted "*rfid counter*").

Table 6.3 shows the domains of the described sensors. To keep the example simple we assume that only a single user is present in the kitchen at any point in time, hence we do not have to distinguish between multiple users.

A central purpose of the sensor model is to represent the conditional dependencies between plan execution states and observed sensor readings. Depending on the principle of operation of a sensor its readings might depend on the executed action and/or the resulting state transition. Figure 6.5 shows a Bayesian network that represents the conditional dependencies that we assume in our example sensor model. The concrete conditional probabilities are then given in Tables 6.4 and 6.5. In these tables, an asterisk '*' in a column denotes that the given probability is independent of the corresponding variable.

The probabilities given in the tables are based on some implicit assumptions about the environment and the general context. For instance we assume that external light in the kitchen is sufficient in 50% of the cases. In the other 50% of the cases we assume an a-priori probability of 0.1 that the light in the kitchen is already turned on. These assumptions result in the reading probabilities given in Table 6.4 for sensor "*wsn light*" in state *idle*. In state v_2 and successor states (excluding v_5 and v_{12}) we then assume that either sufficient natural light is available, or that the user turned on electric light, and hence can exclude the reading *darkness* for sensor "*wsn light*" for these states. As a second example consider the reading probabilities for sensor "*dpm eggs*" in state v_4 (which is part of state set $\mathcal{V}^a$). Here, we assume that 90% of the eggs are equipped with a digital product memory (for eggs without a product memory a *no information* reading is returned). From the eggs with a product memory we assume that 30% are expired. We also represent erroneous sensor readings, e.g. a 3% chance for sensor "*cam stove*" to incorrectly distinguish between good and bad fried eggs.

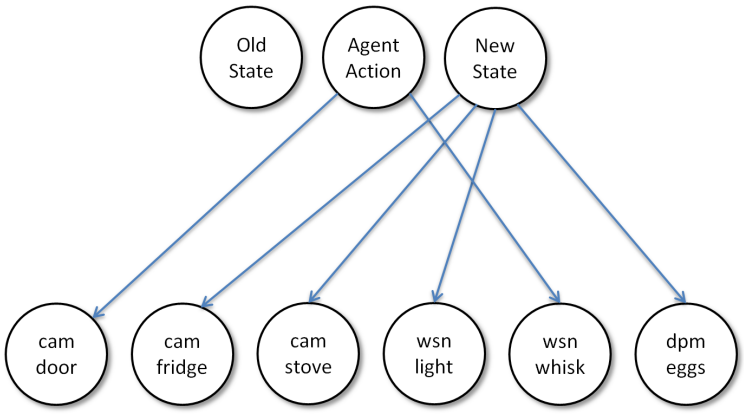


Figure 6.5: Conditional dependencies in the example sensor model.

Sensor s	**Agent Action** α	**End State** v'	**Reading** r	$P_s(*, \alpha, v', r)$
cam door	enter kitchen	*	entered room	1.00
			otherwise	0.00
	leave kitchen	*	left room	1.00
			otherwise	0.00
	otherwise		no movement	1.00
			otherwise	0.00
cam fridge	*	v_3	in front of fridge	1.00
			not in front	0.00
	otherwise		in front of fridge	0.00
			not in front	1.00
cam stove	*	v_{14}	fried egg ok	0.97
			fried egg not ok	0.03
			no fried egg	0.00
	*	v_{10}	fried egg ok	0.03
			fried egg not ok	0.97
			no fried egg	0.00
	otherwise		no fried egg	1.00
			fried egg not ok	0.00
			no fried egg	0.00
wsn light	*	*idle*	natural light	0.50
			electric light	0.05
			darkness	0.45
	*	v_1	natural light	0.00
			electric light	0.00
			darkness	1.00
	*	$\in \{v_5, v_{12}\}$	natural light	0.50
			electric light	0.00
			darkness	0.50
	*	$\in \{v_6, v_{13}\}$	natural light	0.00
			electric light	1.00
			darkness	0.00
	otherwise		natural light	0.50
			electric light	0.50
			darkness	0.00

Note: An asterisk (*) denotes conditional independence

Table 6.4: Probability functions P_s in the example sensor model (part I).

Sensor s	**Agent Action** α	**End State**[1] v'	**Reading** r	$P_s(*, \alpha, v', r)$
wsn whisk	use whisk correctly	*	correct use	1.00
			incorrect use	0.00
			no use	0.00
	use whisk incorrectly	*	correct use	0.00
			incorrect use	1.00
			no use	0.00
	otherwise		correct use	0.00
			incorrect use	0.00
			no use	1.00
dpm eggs	*	$\in \mathcal{V}^a$	eggs ok	0.63
			eggs expired	0.27
			no information	0.10
	*	$\in \mathcal{V}^b$	eggs ok	0.70
			eggs expired	0.00
			no information	0.30
	*	$\in \mathcal{V}^c$	eggs ok	0.00
			eggs expired	0.70
			no information	0.30
	otherwise		eggs ok	0.00
			eggs expired	0.00
			no information	1.00
rfid fridge	*	$\in \mathcal{V}^d$	eggs not in fridge	1.00
			eggs in fridge	0.00
	otherwise		eggs not in fridge	0.10
			eggs in fridge	0.90
rfid counter	*	$\in \mathcal{V}^e$	pan present	0.98
			whisk present	0.00
			neither present	0.02
	*	$\in \mathcal{V}^f$	pan present	0.00
			whisk present	0.98
			neither present	0.02
	otherwise		pan present	0.00
			whisk present	0.00
			neither present	1.00

Note: An asterisk (*) denotes conditional independence

[1] $\mathcal{V}^a = \{v_4, v_7, v_{10}, v_{14}, \ldots, v_{16}\}$, $\mathcal{V}^b = \{v_9, v_{11}, v_{19}, v_{20}\}$, $\mathcal{V}^c = \{v_8, v_{17}, v_{18}\}$, $\mathcal{V}^d = \{v_4, v_7, \ldots, v_{11}, v_{14}, \ldots, v_{20}\}$, $\mathcal{V}^e = \{v_7, v_{10}, v_{14}, \ldots, v_{16}\}$, $\mathcal{V}^f = \{v_8, v_9, v_{11}, v_{17}, \ldots, v_{20}\}$

Table 6.5: Probability functions P_s in the example sensor model (part II).

According to Definition 6.3.1 from the previous subsection our example sensor model can be formally described by the quadruple $(\mathcal{S}, \mathcal{V}, \mathcal{A}, P)$ with

- $\mathcal{S} = \{$cam door, cam fridge, cam stove, wsn light, wsn whisk, dpm eggs, rfid fridge, rfid counter$\}$
- $\mathcal{V} = \{idle, v_1, \ldots, v_{20}\}$
- $\mathcal{A} = \{$enter kitchen, leave kitchen, turn lights on, turn lights off, move to fridge, pick eggs, pick pan, fry eggs, stir eggs, pick whisk, use whisk correctly, use whisk incorrectly, dispose eggs, other, idle$\}$
- The set of probability functions P is given in Tables 6.4 and 6.5

6.4 State-Aware Plan Recognition System

In order to perform plan recognition, one has to derive the high-level plans from low-level sensor observations. For this purpose, plan library and sensor model have to be merged to a single *combined plan recognition model*. This model can then be used to derive a *plan recognition function* $pr(h, o)$, which computes one iteration of the plan recognition process by updating the current hypothesis h based on the current observation o[4].

A given plan library and sensor model can only be merged to a combined plan recognition model if their domains are compatible. "Compatibility" in this context means, that the sensor model "knows" at least all plan execution states and agent actions which are part of the considered plan library. This is not self-evident: Plan library and sensor model are typically defined independently of each other. While the plan library is usually agent-dependent (as the given probabilities reflect the agent's practical knowledge and personal preferences regarding certain plans, plan variants, and basic actions), the sensor model typically depends on the concrete implementation of the environment and the applied sensors, and thus generally is defined individually for each environment. Compatibility between a plan library and a sensor model is formally defined in Definition 6.4.1.

If a concrete plan library and a concrete sensor model are compatible, the combined plan recognition model can be represented as a dynamic Bayesian network (DBN, see section 4.3) with the structure shown in Figure 6.6. This network is constructed by merging the plan selection and execution DBN (see Figure 6.3) with the according sensor model network (see Figure 6.4). The compatibility criterion given in Definition 6.4.1 ensures that the domains of the state and action variables match the conditional probabilities of the

[4]An observations is a set of sensor readings, which for each active sensors contains exactly one reading.

Definition 6.4.1. A plan library $(\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ and a sensor model $(\mathcal{S}, \mathcal{V}', \mathcal{A}', P)$ are called **compatible**, if

- $\mathcal{V} \subseteq \mathcal{V}'$, and
- $\mathcal{A} \subseteq \mathcal{A}'$

sensor variables. States and actions which are introduced in the sensor model but are not used in the plan library are assumed to occur with a fixed a-priori probability of 0.

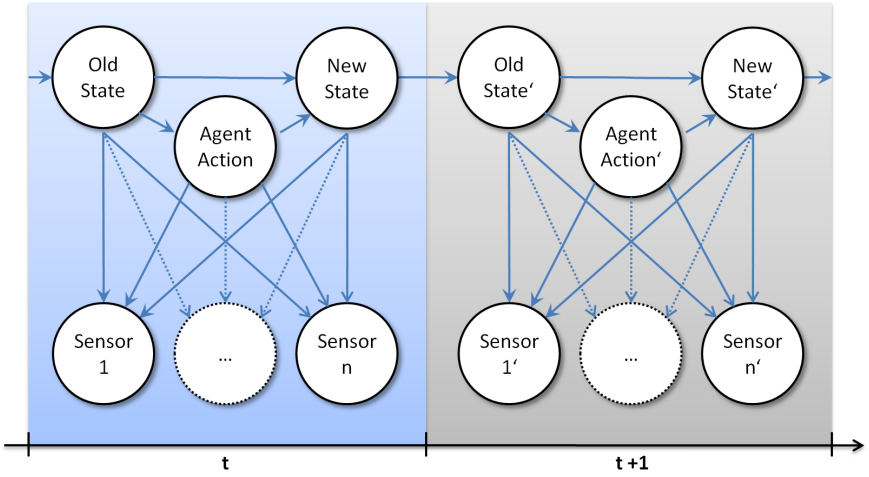


Figure 6.6: The combined dynamic Bayesian plan recognition network is constructed by melting a plan library DBN with a compatible sensor model network.

After the combined plan recognition network has been constructed, it can be used to perform one iteration of the actual plan recognition process as described by the following 3-step algorithm:

1. The *Old State* node's a-priori distribution is set to the current plan recognition hypothesis (the initial plan hypothesis is $h(idle) = 1$ and $h(v \neq idle) = 0$).

2. Observation data (if available) is assigned as evidence to the according nodes in the sensor model subnetwork.

3. After performing probabilistic inference on the resulting DBN, the updated plan hypothesis can be queried from the *New State* node.

This algorithm can be formulated as plan recognition function $pr_{L,M}(h, o)$:

$$pr_{L,M}(h, o) = (v' \mapsto \sum_{v \in \mathcal{V}} h(v) Pr^*_{L,M}(v'|v, o)) \tag{6.1}$$

where the conditional probability $Pr^*_{L,M}(v'|v, o)$ is the probability that a transition to v' occurs given previous state v and observations o. $Pr^*_{L,M}$ is calculated from the combined dynamic Bayesian plan recognition network for plan library L and sensor model M. In the following the indexes L and M will be omitted if the scope of $pr_{L,M}$ is clear from the context. We can now apply our existing definition of plan recognition systems (see Definition 3.2.1) to define the plan recognition system using our state-based approach as follows:

Definition 6.4.2. A **state-based plan recognition system** is a plan recognition system $(L, M, O_{\mathcal{S}}, \mathcal{H}_L, h_0, pr_{L,M})$ where

- $L = (\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ is a plan library according to Definition 6.2.2
- $M = (\mathcal{S}, \mathcal{V}', \mathcal{A}', P)$ is a sensor model according to Definition 6.3.1
- L and M are compatible
- $O_{\mathcal{S}}$ is the power set of all readings eventually returned by $\mathcal{S}$
- $\mathcal{H}_L$ is the set of all state-aware plan hypotheses regarding L
- h_0 is the initial hypothesis with $h_0(idle) = 1$ and $h_0(v \neq idle) = 0$
- $pr_{L,M}$ is the plan recognition function described above

In our example from the kitchen domain the combined plan recognition network is constructed by merging the plan selection and execution DBN that represents the sample plan library from subsection 6.2.2 with the sample sensor model network shown in Figure 6.5.

In the following we apply the resulting network to demonstrate one iteration of plan recognition in our sample domain starting with the initial hypothesis h_0. The resulting probabilities for different exemplary observations are listed in Table 6.6. After the *Old State* node's a-priori probabilities

Observations o	$pr(h_0, o)(idle)$	$pr(h_0, o)(v_1)$	$pr(h_0, o)(v_2)$
{}	0.800	0.090	0.110
{natural light}	0.879	0.000	0.121
{electric light}	0.421	0.000	0.579
{darkness}	0.800	0.200	0.000
{no movement, electric light}	1.000	0.000	0.000
{entered room}	0.000	0.450	0.550
{entered room, natural light}	0.000	0.000	1.000
{entered room, electric light}	0.000	0.000	1.000
{entered room, darkness}	0.000	1.000	0.000

Table 6.6: Hypotheses for different observations after one iteration of plan recognition starting from the initial hypothesis h_0 (rounded).

have been set according to the initial hypothesis' probability distribution, we iteratively assign the observed sensor readings (first column) as evidence to the associated sensor nodes. Readings might only be provided for a subset of sensors, e.g. because all other sensors have been deactivated. For sensors without an observed reading no evidence is assigned to the according nodes. For each observation we then perform probabilistic inference to calculate the value of plan recognition function *pr* respectively the successor hypothesis. For each considered observation Table 6.6 lists the resulting probability values for states *idle*, v_1, and v_2 (second to fourth column).

The numbers in the table provide interesting insight into the general reasoning of the plan recognition system. For instance, in the absence of evidence from the *cam door* sensor the observation of electric light in the kitchen is interpreted by the system as evidence for the presence of the user, as the a-priori probability of electric light being turned on without the presence of the user with 0.05 is rather low (see row *wsn light* for state *idle* in Table 6.4).

On the other hand, we can see that the readings of the *wsn light* sensor help distinguishing whether the user is in state v_1 or v_2, and hence predicting whether the user will turn on the light in the next step or not (see Figure 6.2). More sophisticated examples for the prediction of actions will be given in the following subsection.

6.5 Predicting Goals and Future Actions

As introduced earlier, one of the main motivations for the application of plan recognition techniques is to learn about the agent's intended goals and to anticipate the agent's subsequent actions. This section describes how to infer

such information from a given plan library and the current plan hypothesis by using the plan recognition function to predict future hypotheses.

The process of predicting goals and future behavior is tightly connected with the process of explaining past behavior. While in the explanation case plan hypotheses are constructed based on past observations, the prediction case requires the construction of (expected) plan hypotheses based on yet unknown observations. While in the first case one uses a plan recognition function *pr* to derive the current hypothesis from an old hypothesis, in the second case one can use the same function *pr* to derive the expected upcoming plan hypothesis from the current plan hypothesis. The lack of observation information in the prediction case is accounted for by providing *pr* with an empty set of observations. In this case, the new hypothesis is calculated solely based on the *a-priori probabilities* respectively *conditional probabilities* given in the plan library. As it is assumed that the plan library is a suitable description of the agent's plan selection and execution process, this approach provides the best possible estimate on upcoming hypothesis.

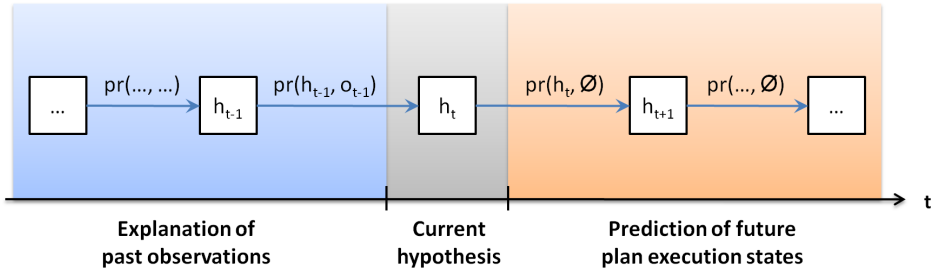


Figure 6.7: A plan recognition function *pr* can be used for the derivation of the current plan hypothesis (which explains a sequence of previous observations $[o_0, \ldots, o_{t-1}]$) as well as for the anticipation of future plan execution states (in the form of a sequence of expected plan hypotheses $[h_{t+1}, \ldots, h_{t+n}]$).

Figure 6.7 graphically illustrates the two use cases of a plan recognition function *pr* that have been discussed above: On the one hand, *pr* is used to retrospectively infer the current plan execution hypothesis h_t from the preceding hypothesis h_{t-1} and the most recent set of observations o_{t-1}, which are used as evidence to narrow down the set of potential candidate states. On the other hand, *pr* can be used to anticipate future plan execution states from the current hypothesis based on an empty (because yet unknown) set of observations. One iteration of *pr* on the current plan recognition hypothesis h_t provides a prediction on the expected plan hypothesis h_{t+1} in the immediately following time step. This process of prediction can be iteratively

repeated to anticipate the expected plan hypotheses at any arbitrary future point in time.

In order to predict the agent's intended goal (or more exactly in order to identify the probabilities of possible candidate goals), we have to repeat the process of prediction until all evidence has accumulated at terminal states. The resulting hypothesis is called a *terminal hypothesis*, and is formally defined as follows:

Definition 6.5.1. A **terminal hypothesis** regarding some plan library $L = (\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ is a plan recognition hypothesis h_L which meets the condition

$$\sum_{g \in \mathcal{G}} h_L(g) = 1$$

Note that we can only be sure that *all* evidence sooner or later accumulates at terminal states (and thus the termination criterion given in Definition 6.5.1 is met) if the plan library is acyclic[5]. Otherwise, the library might contain plans of infinite length, which might cause our termination criterion to fail. In such cases, we have to use some other termination criterion here. A trivial approach is to *limit the maximum number of iterations*. More sophisticated approaches might use techniques for *cycle detection*, like the "tortoise and hare" algorithm [Flo67].

We now give a function $epg : \mathcal{H} \times \mathcal{G} \mapsto [0, 1]$, which uses iterative hypothesis prediction to calculate the *expected probability of reaching goal g* based on a given (current) plan hypothesis h:

$$epg(h, g) = \begin{cases} h(g) & \text{if } h \text{ is a terminal hypothesis} \\ epg(pr(h, \emptyset), g) & \text{otherwise} \end{cases}$$

For an example consider the sample models introduced in the previous sections. Assume that the current hypothesis h_t states that the user is in v_8 with probability 0.2 and in v_9 with probability 0.8. We now apply function epg to calculate the probability that the user reaches v_{19}. In order to compute $epg(h_t, v_{19}) = 0.64$ two recursion steps are required. After the first recursion

[5] A plan library is acyclic, if for each possible partial transition sequence $[v_1, \ldots, v_n]$ the condition $v_1 \neq v_n$ holds. The plan graph of an acyclic plan libraries itself always is acyclic.

we get $h_{t+1} = \{v_{11} : 0.20, v_{17} : 0.16, v_{18} : 0.04, v_{19} : 0.60\}$[6]. After the second recursion step we get $h_{t+2} = \{v_{17} : 0.16, v_{18} : 0.04, v_{19} : 0.64, v_{20} : 0.16\}$. As h_{t+2} is a terminal hypothesis the recursion ends at this point.

Besides the expected probability of reaching a certain goal, we might also be interested in the expected probability of the occurrence of certain future actions. Similar to function *epg*, we can define a function $epa : \mathcal{H} \times \mathcal{A} \mapsto [0, 1]$, which returns the probability that an action α occurs in the next time step given some (current) hypothesis h. The probability of the occurrence of action α can be directly computed from the underlying library's action selection function s:

$$epa(h, \alpha) = \sum_{v \in \mathcal{V}} h(v)s(v, \alpha)$$

For an example we again assume that the current hypothesis is $h_t = \{v_8 : 0.2, v_9 : 0.8\}$. Then we can compute the probability of action *use whisk correctly* in the next time step by:

$$epa(h_t, \text{use whisk correctly}) = 0.2 \cdot 0.6 + 0.8 \cdot 0.75 = 0.72$$

One-Shot vs. Continuous Plan Execution

Recall from section 6.2, that we assumed so far, that the agent's plan execution ends as soon as a terminal plan execution state is reached. In the following this behavior is called *one-shot plan execution*. In practical applications, the agent most often executes more than one plan in a row. This behavior in the following is called *continuous plan execution*. One-shot and continuous plan execution are characterized and distinguished as follows:

- **One-Shot Plan Execution:** In the case of one-shot plan execution we assume that the agent executes exactly one plan instance. This implies that the agent stops plan execution as soon as a terminal state is reached.

- **Continuous Plan Execution:** In the case of continuous plan execution we assume that the agent executes a (possibly open-ended) sequence of plans. This implies that after reaching a terminal state the agent's plan execution process starts over in the *idle* state.

[6]Expression $h_t = \{v_1 : p_1, \ldots, v_n : p_n\}$ is a shorthand for $h_t(v_i) = p_i$ for $1 \leq i \leq n$ and $h_t(\_) = 0$ otherwise.

The above mentioned method to anticipate expected plan hypotheses is only valid in the case of one-shot plan execution, as one-shot execution was explicitly assumed when the plan library DBN and thus our plan recognition function *pr* (via the combined plan recognition DBN) was defined in section 6.2 respectively section 6.4. To account for the case of continuous plan execution, we have to take special care for plan recognition hypotheses which assign evidence to terminal states. As the termination of the current plan might imply the start of a new plan, this evidence has to be considered as evidence for the *idle* state in the next prediction iteration.

Formally, this case can be handled by the following *"continuouization" function* $c : \mathcal{H} \mapsto \mathcal{H}$, which derives a new plan recognition hypothesis from a given plan recognition hypothesis by "transferring" all evidence on terminal states to the *idle* state:

$$c(h) = (v' \mapsto \begin{cases} h(idle) + \sum_{g \in \mathcal{G}} h(g) & \text{if } v' = idle \\ 0 & \text{if } v' \in \mathcal{G} \\ h(v') & \text{otherwise} \end{cases} \quad)$$

In order to anticipate expected plan recognition hypotheses in the continuous plan execution case, one has to apply this continuouization function after each prediction step. This can be formalized by the following function $cp : \mathcal{H} \times \mathbb{N}^+ \mapsto \mathcal{H}$, which recursively anticipates the expected plan hypothesis in the n-th next time slice based on some current plan recognition hypothesis h in the continuous plan execution case:

$$cp(h, n) = \begin{cases} c(h) & \text{if } n = 0 \\ c(pr(cp(h, n-1), \emptyset)) & \text{otherwise} \end{cases}$$

By using function *cp* with the current hypothesis h and assuming continuous plan execution, we can now estimate the expected probability that action α will be executed in the n-th next state from now through $epa(cp(h, n), \alpha)$, and similarly estimate the expected probability that some goal state g is intended in the n-th next state from now through $epg(cp(h, n), g)$.

Note that the expected probability of goals is only defined for a single plan, and not for (possibly infinite) sequences of independent plans. Thus, function *epg* always considers the given hypothesis as state description of a one-shot plan.

6.6 Summary

In this chapter we presented a state-aware plan recognition approach which addresses the problem of plan recognition based on state observations in instrumented environments (see section 1.1). The presented approach provides the foundation for the work that is presented in the rest of this thesis. In particular, the state-aware plan recognition approach is used as follows: It provides the theoretical foundation for the utility model for observation information in plan recognition applications, which we develop in chapter 7. An implementation of our state-aware plan recognition approach realizes the plan recognition component in the resource-aware plan recognition toolkit RePReTo that we describe in chapter 8. The RePReTo toolkit is practically applied in our *Smart Kitchen* environment, which we present in chapter 9. The performance of the proposed plan recognition approach (among other aspects) is evaluated in chapter 10.

7

A Decision-Theoretic Utility Model for Observation Information in Plan Recognition Applications

A crucial factor for the successful application of plan recognition is the availability of meaningful observation information. The more evidence about the agent's behavior and the state of the environment is available to the plan recognition system, the more accurate the system's hypotheses can be. On the other side, it can be reasonable or even required to restrict the set of queried information sources due to existing resource constraints (see section 2.3). In order to decide on the most promising subset of information sources to use, one then needs a way to judge and compare the utility of information provided by different sets of sources.

This chapter proposes a *decision-theoretic utility model* for observation information that allows solving the problem of sensor selection in plan recognition applications. At first we discuss the nature of the utility of observation information in plan recognition application (see section 7.1). Next we derive a utility model that is based on *Partially Observable Markov Decision Processes* (POMDPs) (see section 7.2). We revise our plan recognition model from chapter 6 to account for the additional assumptions that we introduced in order to formulate the proposed utility model (see section 7.3), and conclude with a presentation of two sensor selection strategies which make use of the proposed utility model to solve the problem of sensor selection in plan recognition applications (see section 7.4).

7.1 Defining Observation Information Utility

The goal of the utility model that is introduced in this chapter is to provide a quantitative measure of the usefulness of observation information in practical

plan recognition applications. This measure should be grounded on a well-defined theoretical framework, which is computational feasible yet powerful enough to account for all relevant factors which might influence the value of observation information.

This section starts with a discussion of intrinsic versus extrinsic observation utility and motivates why it is essential to include the utilization of plan recognition information through higher-order support systems in any consideration regarding observation utility in plan recognition applications. Next, we propose to apply decision theory as a suitable theoretical framework for the definition of extrinsic observation utility. We develop the general idea of a decision-theoretic utility measure for observation information and provide a formal high-level definition which then is refined in later sections.

7.1.1 Intrinsic vs. Extrinsic Utility

The most obvious benefit of observation information is to serve as additional evidence in the plan recognition system's reasoning, and thus to positively influence the accuracy of the resulting hypothesis. In the following, we call this kind of utility the immediate, or *intrinsic utility* of observation information. A naïve definition of observation information utility might be based on intrinsic utility, where the utility of an observation negatively correlates with the decrease in hypothesis uncertainty that results from the inclusion of this information into the plan recognition system's reasoning process. However, the following thought experiment reveals that intrinsic utility often is not a good measure for observation utility in plan recognition applications:

> Imagine that a plan recognition system is following the actions of an elderly user John while preparing a birthday cake for his friend Tim. The system's hypotheses are used by a support application that warns John if he uses ingredients that possibly threaten the health of John or other people. Assume that the plan recognition system so far has discovered that John is most probably going to prepare a cake for Tim, and further knows that Tim is allergic to nuts. However the system is unsure whether John plans to prepare a nut cake, a chocolate cake, or some other kind of cake without chocolate or nuts. Now the system has to decide whether to look for an addition of nuts, or an addition of chocolate.

With respect to intrinsic utility, both observations are equally valuable, as both – if actually observed – would allow narrowing down the plan hypothesis to a single plan in the next plan recognition iteration. However, to know

whether nuts are added or not is practically more important in this case, as it would allow the external support system to reliably decide whether to issue an allergy warning about nuts to the user or not.

The thought experiment reveals, that in order to assess the utility of observation information in practical plan recognition applications, we have to explicitly consider how plan recognition information is utilized by the applied higher-order support system. In the example above, the true utility of observation information lies in its ability to indirectly enable the provision of appropriate support through the external system. Accordingly, this kind of utility is called indirect or *extrinsic utility* of observation information. Here, the utility of an observation correlates with the increase in added value, that the support system can "generate" based on the plan recognition results. Additional observation information indirectly increases the expected added value by increasing the quality of the hypothesis and thus providing the support system with a better foundation for choosing its actions.

The definitions of intrinsic and extrinsic utility correspond to the cases of intrinsically and extrinsically motivated plan recognition:

- **Intrinsically Motivated Plan Recognition**: In the case of intrinsic motivation, plan recognition is primarily performed for the sake of its own. The main objective is to learn as much about the agent's plans and intentions as possible. This implies that every plan is equally important or unimportant to be recognized. Accordingly, the correct recognition of any arbitrary plan would have to be considered equally valuable by a utility model which assumes intrinsic motivation.

- **Extrinsically Motivated Plan Recognition**: Extrinsically motivated plan recognition is primarily performed to provide hypotheses about the currently executed plan to some higher-order support system, which then uses this information to trigger the execution of support actions. The main objective here is to maximize the added value that is generated by the support system's actions. In this case, certain plans might be more important to recognize than others, e.g. because the support that can be offered on the recognition of a plan that threatens the health of the agent is more valuable or important than the support that can be provided on the recognition of a plan that only affects the agent's convenience. A utility model which assumes extrinsic motivation would have to take such aspects into account, and accordingly would have to weigh the importance of observations according to the relevance of the support which they enable.

The specific characteristics of intrinsic and extrinsic utility for plan recognition and the resulting consequences for the definition of the utility of observation information in plan recognition applications are summarized in Table 7.1. While it helps to distinguish between intrinsic and extrinsic utility in order to understand the general idea of the proposed utility measure for observation information, intrinsic utility in fact is a *special case* of extrinsic utility, in which all plans are equally important. As in practical applications this special case rarely applies, the utility model that we develop in the rest of this chapter considers the more general case of extrinsic observation information utility.

	Intrinsic Utility	**Extrinsic Utility**
Purpose of Plan Recognition	not considered	input for decision-making
Topmost Goal	accurate recognition	generation of added value
Recognition Preference	all plans equally	"important" plans preferred
Utility Measure	recognition rate	added value of support
Practical Relevance	low	high

Table 7.1: Comparing intrinsic and more general extrinsic observation information utility.

7.1.2 Decision-Theoretic Measure for Extrinsic Utility

As motivated in the previous section, the main purpose of additional information in the case of extrinsic motivation is to support the external system's decision making by improving the accuracy of the resulting plan hypothesis, thus leading to the provision of "better suited" and more valuable support. A formal utility model therefore has to consider the support system's decision process and in particular has to focus on the following dependencies:

1. The influence of observation information on the resulting hypothesis.
2. The influence of the resulting hypothesis on the provided support.
3. The influence of the provided support on the added value.

A theory that was specifically developed for the purpose of reasoning about decisions, expected outcome utilities, and associated uncertainties is *decision theory* (see section 4.4). Decision theory provides us with a definition of the *expected utility of information*, which shows several similarities to the intuitive understanding of information utility that was drafted above:

According to decision theory, an information item is of no specific value, unless it is "used" by a decision maker in a decision situation to make "better" decisions. Given a certain decision situation, the decision-theoretic utility of an information item then is defined as the amount of money (or some other arbitrary but fixed utility measure[1]) that the decision maker is willing to pay for knowing this information in advance of making the decision.

Decision theory furthermore states, that for a *rational risk-neutral decision maker* the fair price of some information item equals the increase in expected utility of the associated decision situation that results from knowing the considered information before making the decision.

In the case of extrinsically motivated plan recognition, the relevant decision problem is for the support component to choose the best support action given the current plan hypothesis. We call the resulting decision problem the *support decision problem* (SDP). SDP will be formally introduced in the following section. For now, we assume that we are given a function $u_{SDP}(\beta, v)$ which provides us with the utility of executing support action β in state v, and a function $B_{SDP}(h)$ which provides us with the solution (the best action to execute, see Definition 4.4.3) of decision problem SDP for hypothesis h.

Recall from section 4.4 that in order to calculate the expected utility given incomplete information we have to sum up the resulting utilities for all possible situations while weighting each utility by the probability of the associated outcome situation. In the case of an observation source, the relevant situations are the possible state transitions in the plan graph and the resulting observations. The resulting probabilities of transitions and observations can be computed from the current plan hypothesis through the probability distributions given by the assumed plan library and sensor model.

For each possible transition and each possible resulting observation we then anticipate the reasoning of the plan recognition system by applying plan recognition function pr on the current hypothesis and assumed observation. Then, we solve the support decision problem two times: Once for the resulting anticipated plan hypothesis, and again for the hypothesis obtained in the uninformed case (from an empty observation set). We then compare the utilities of both actions in the assumed outcome state[2].

[1] For reasons of simplicity we assume in the following that utility is always measured in a money-equivalent unit.

[2] When computing the outcome utility we iterate over all possible states and hence for each iteration "know" the true resulting state. However, this information is not used in the anticipation of the updated plan hypothesis through function pr, as this information will not be available to the plan recognition system when sensing and reasoning finally is performed in the real environment.

Definition 7.1.1. The **expected utility of an observation information source** s given hypothesis h is defined as

$$EUS_s(h) = \sum_{v \in \mathcal{V}, \alpha \in \mathcal{A}, v' \in \mathcal{V}, o \in O_S} h(v)\, p(v, \alpha, v', o)\, (u_{SDP}(\beta_1, v') - u_{SDP}(\beta_0, v'))$$

where

- $p(v, \alpha, v', o) = s(v, \alpha)t(v, \alpha, v')p_S(v, \alpha, v', o)$ is the situation probability function (functions s and t are given by the assumed plan library and function p_S is given by the assumed sensor model)
- $\beta_1 = B_{SDP}(pr(o, h))$ is the action chosen in the informed case
- $\beta_0 = B_{SDP}(pr(\emptyset, h))$ is the action chosen in the uninformed case

7.1.3 The Support Decision Problem

The previous section already provided an intuitive understanding of the support decision problem SDP, which serves as the decision-theoretic foundation for the proposed utility model for observation information. In this section we discuss this decision problem in more detail. In order to formally define SDP, we assume that the higher-order support system has available a set of support actions (called the *support action repertoire*), from which it chooses exactly one action in each time step. By choosing a special obligatory null action *none*, the support system may decide to stay inactive and provide no support in a time step.

We further assume that a function u_{SDP} exists, which for each plan execution state and system action provides the expected *long-term utility* of executing a given action in a particular state. Long-term utility here includes the direct (or short-term) utility that is immediately realized by the execution of a support action, and the utility indirectly caused by this action in later steps of the executed plan, e.g. through the final attainment of a higher-valued goal (a more detailed description of long-term vs. short-term utility is given later in this chapter). The support decision problem for a known plan recognition hypothesis then is to choose a support action with maximum expected long-term utility. Formally, the support decision problem SDP is a special case of the general decision problem (see Definition 4.4.1) and defined as follows:

Definition 7.1.2. The **support decision problem** SDP is a decision problem $(\mathcal{B}, \mathcal{V}, u_{SDP})$, where

- $\mathcal{B}$ is the set of the system's *support action repertoire*.
- $none \in \mathcal{B}$.
- $\mathcal{V}$ is a finite set of *plan execution states*.
- $u_{SDP} : \mathcal{B} \times \mathcal{V} \mapsto \mathbb{R}$ is the SDP utility function, where $u_{SDP}(\beta, v)$ returns the long-term utility of executing support action β in plan execution state v.

The key question now is how to define function u_{SDP}. As motivated in section 7.1, utility should be related to the value that the owner of the system experiences through the provision of support. In the following we assume that the outcome value of plan execution is determined by two factors: The costs associated with the execution of actions which realize a plan, and the rewards associated with the attainment of (partial) goals that are connected with the executed plan. Through the provision of support, the support system can improve the resulting outcome value by influencing the observed agent's plan selection and execution process in a way that results in lower costs and/or higher rewards.

Recall from section section 4.4 that a decision problem can be represented as an influence diagram. Figure 7.1 shows the influence diagram for SDP which illustrates the general idea drafted above. The diagram is based on the plan recognition DBN (see Figure 6.6) that we introduced in section 6.4 (for reasons of clarity, the set of individual sensor nodes has been replaced by a single *Sensors* node).

The influence diagram adds two additional nodes to each time slice: The *Support Action* node is a decision node which represents the support system's choice regarding the support action to execute. The domain of this node equals the system's *support action repertoire*. The second additional node is the *Utility* node, which defines the immediate utility resulting from the considered time slice. As motivated above the resulting utility depends on the cost of the executed action and a possible reward associated with the reached (intermediate) goal possibly connected with the reached plan execution state. As a third factor, we have to additionally account for the costs that are associated with the executed support action. These dependen-

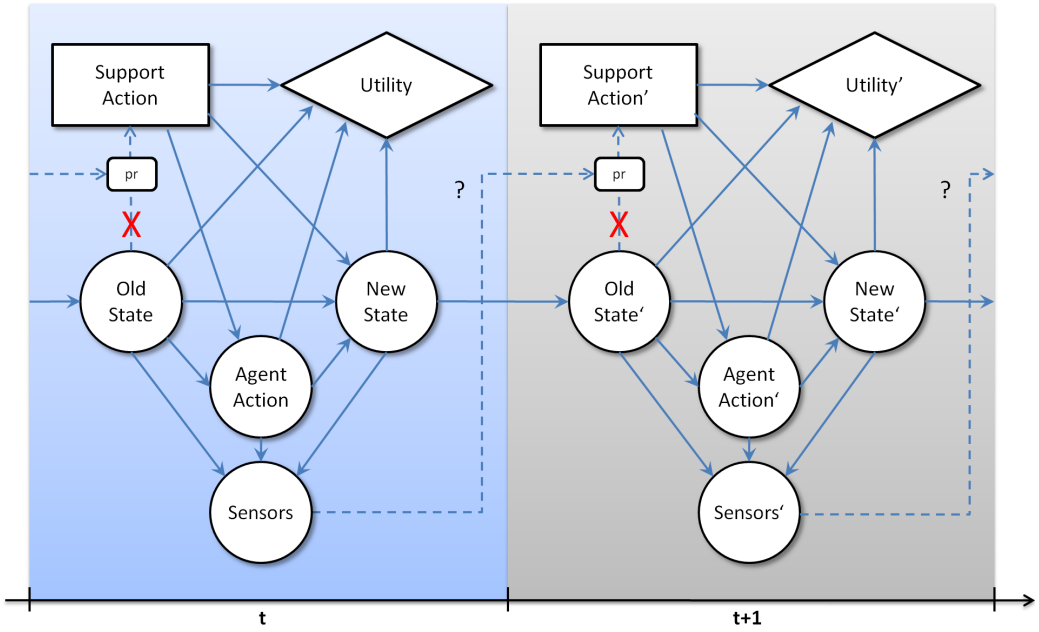


Figure 7.1: Influence diagram showing the causal relationships that exist within the support decision problem (SDP). The diagram is based on the combined plan recognition DBN (see Figure 6.6).

cies are represented by the three arcs that point towards the *Utility* node. The influence of the chosen support action on the agent's plan selection and execution process is modeled by the two additional edges from the *Support Action* node to the *Agent Action* and *New State* nodes.

The influence diagram given in Figure 7.1 further shows, that the choice of the optimal support action depends on the current plan execution state (represented by the value of the *Old State* variable). The true current plan executing state is hidden to the system. Instead, an estimation of the plan execution state has to be derived by the system from the previous plan hypothesis and sensor observation information via plan recognition function *pr*. This fact is represented by the dashed arcs between the nodes *Sensors*, *Old State*, and *Support Action*. These arcs reflect the use of plan recognition function *pr* in the formal definition of the expected utility of observation information (see Definition 7.1.1) that we presented in the previous section.

Given the influence diagram, we can now principally (if we know the concrete conditional probabilities and resulting utility values) decide on the support action which results in the best expected utility in the current time slice. However, optimizing short-term utility does not necessarily result in

a maximum outcome utility for the overall plan. The reason for this is that SDP is a *sequential decision problem*, which means that the eventually resulting *long-term utility* does not only depend on the decision made in the current time step, but also on decisions that have to be made in future time steps (in our case decisions on support that should be provided during the remainder of the executed plan). Thus, an option that is suboptimal in the current time slice might allow for the provision of even more valuable support in the future, and thus in the long run can be the better decision alternative. This is the reason why we required function u_{SDP} in Definition 7.1.1 to provide the long-term utility of executed actions.

Mature approaches exist to calculate long-term utilities in a sequential decision problem if the decision problem can be formulated as a *partially observable Markov decision process* (POMDP). In the following section we explain how SDP can be formulated as a POMDP, which then allows us to apply existing standard algorithms to solve SDP, calculate EU_{SDP}, and finally compute the expected utility of observation information as described in Definition 7.1.1.

7.2 Solving the Support Decision Problem

The previous section proposed a utility measure which defines the utility of observation information by means of the maximum expected utility $\widehat{EU}_{SDP}$ of the support decision problem SDP. In order to solve SDP and to calculate $\widehat{EU}_{SDP}$, we have to face the following three subproblems:

At first, we have to formally model the dependencies introduced by the influence diagram shown in Figure 7.1. At second, we have to describe how to calculate for each action and plan hypothesis the expected long-term utility from the individual time slices' immediate utilities (as given by the *Utility* nodes). At last, we have to efficiently search for the support action which maximizes the expected utility and thus provides the solution of SDP.

The first subproblem is addressed by the introduction of two new models. The *support model* describes the influence of the support actions on the agent's plan selection and execution process (see Figure 7.2, left)) and is discussed in more detail in subsection 7.2.1. The *cost-reward model* describes the immediate utility that results from one time slice through the execution of execution of actions and the resulting state transitions (see Figure 7.2, right) and is discussed in more detail in subsection 7.2.2.

The second and third subproblems are addressed by using the previously defined support and cost-reward models to formulate SDP as a *partially observable Markov decision process* (POMDP). POMDPs have been developed

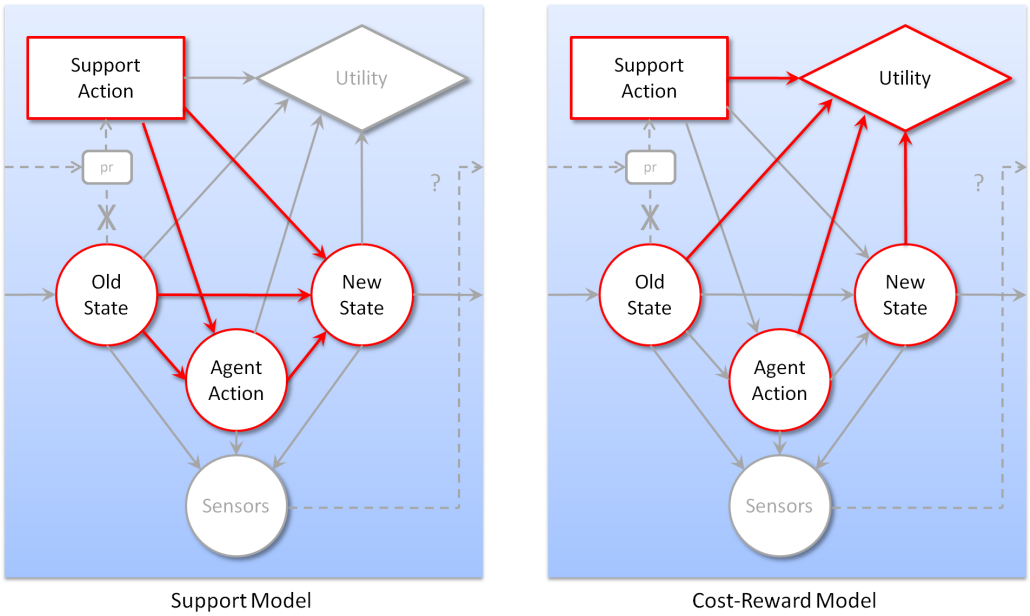


Figure 7.2: The *support model* describes the influence of support actions on the agent's plan selection and execution process (left). The *cost-reward model* describes the immediate utility of the execution of support actions (right).

to model sequential decision making in partially observable domains. The resulting SDP POMDP can then be solved by using one of several existing optimal and approximating standard algorithms. The resulting POMDP policy can then be efficiently evaluated in order to solve SDP for a given plan hypothesis. The process of constructing and solving the SDP POMDP is described in more detail in subsection 7.2.3.

7.2.1 Support Model

The support model formally describes the influence of the higher-order support system's actions on the observed agent's plan selection and execution process. A support system has two general possibilities to influence the agent's behavior: Either directly through the active manipulation of the environment, or indirectly through the provision of information which might passively influence the user's behavior.

Direct support actions can be further subdivided into assistance and intervention actions, where former actions try to actively support the agent in executing the chosen plan, and latter actions try to actively prevent the agent from executing the chosen plan.

Indirect support influences the agent's decision making during plan selection and execution, and thus can be used for assistance as well as for intervention. However, the agent is free to ignore indirect support, while it might not be able to influence the results that originate from direct support. In the following, the three main classes of support actions are discussed in more detail. This discussion later will help developing the formal support model:

- **Active Anticipation**: Active anticipation comprises all actions which actively influence the state of the environment or the behavior of the user in a way that anticipates the intended results of the agent's next action, e.g. through the execution of actions on behalf of the agent. Thereby, direct assistance does not try to change the executed plan itself. In the direct assistance case, added value is generated by the fact that the execution of a certain action might be "cheaper" (takes less time, consumes less cognitive resources, costs less money, etc.) for the system then it would be for the agent. On the downside, direct assistance might lead to undesirable results if executed in the wrong state.

- **Active Intervention**: Active intervention comprises all actions which actively influence the state of the environment or the behavior of the user in a way that prevents the agent from following a costly plan or from reaching a dangerous state. To reach this goal, active intervention aims at enforcing a change of the current plan, e.g. through the execution of actions that result in transitions to more desired states[3]. In the direct intervention case, added value is generated by the fact, that the agent is prevented from executing high cost or low reward plans. Direct intervention actions might lead to undesirable results if executed in the wrong state.

- **Passive Decision Support**: Instead of actively influencing the agent's environment or its actions, a support system can passively influence the agent's behavior (in particular its plan selection process) through the provision of relevant information (and hence decision support), which then allows the agent to act more efficiently. The system might for instance offer support by reminding the agent on upcoming actions, pointing out the expected outcomes of different action/plan alternatives, or propose better/more efficient actions or plans. Like in the

[3]Recall from chapter 3 that for each "perfect" plan the plan library might additionally contain several plan alternatives which describe suboptimal or incorrect realizations of the "perfect" plan.

case of direct support, the provision of inappropriate information might lead to undesirable results. However, in contrast to direct support, the agent might choose to ignore provided passive support if it can identify the support as inappropriate.

In order to model the influence of support actions on the agent's plan selection and execution process we shortly review how plan selection and execution have been modeled for our state-based plan recognition approach.Recall from section 6.2 that a plan library describes the agent's behavior by means of the agent's action repertoire, the plan execution state space, and the action selection and transition functions, which determine how the agent moves through the plan execution state space during plan execution.

If executed in a "suitable" state, support actions might influence the behavior of the agent by either affecting the agent's action selection process (this primarily holds for passive support actions), or the resulting state transitions (this primarily holds for active assistance and intervention actions). Both cases can be understood as local deviations from the "regular" plan library that are caused by the execution of support actions. The purpose of the support model is to describe these deviations.

Figure 7.3 illustrates the general idea of the support model in the context of the previously introduced cooking example: The foundation of the support model is the agent's plan library, which describes the case where no support is provided by the support system (represented by system action *none*). In Figure 7.3 this case is illustrated by the gray underlay of the sample plan library graph from subsection 6.2.2.

Besides the obligatory *none* action, our sample support model introduces five system actions $\{\beta_1, \ldots, \beta_5\}$ which are marked by individual colors in Figure 7.3. The system actions support the user in different ways:

- *System action* β_1 (green) provides an example of *active anticipation* by remotely turning on the lights in the kitchen on behalf of the user. β_1 is only useful to apply if the user entered the room in darkness (as represented by state v_1). If executed β_1 generates added value by reducing the user's effort thus increasing her comfort. As the effect of user action *turn lights on* is now achieved through system action β_1, the user is assumed to "perform" action *idle* in response.

- *System action* β_2 (red) provides an example of *active intervention* by remotely turning off the lights in the kitchen if the user leaves the kitchen with the lights on (which is represented by state v_6). In this case added value is generated by reducing energy costs, which is represented

by a higher reward of the goal *out of kitchen with lights off* compared to goal *out of kitchen with lights on*. Again the user is not actively involved, which again is represented by user action *idle*.

- *System actions* β_3 (orange), β_4 (blue), and β_5 (purple) are examples of *passive decision support*. These actions provide assistance which may influence the user's plan selection and execution process. β_3 proposes the user to change her plan to *have scrambled eggs* if a problem occurs during the preparation of fried eggs (which is represented by state v_{10}). The support model assumes that the user follows this advice with a probability of 0.95. β_4 proposes to dispose expired eggs (v_8) if the plan is to prepare a mayonnaise. The model assumes that the provision of this support increases the probability of a *dispose eggs* action to 0.8 in the supported case. The same effect appears if β_4 is erroneous executed in the case of good eggs (v_9). β_5 finally explains the user how to correctly apply the whisk during the preparation of mayonnaise (here state v_{11} represents the erroneous case), and increases the probability that the user notices and corrects her error from 0.2 to 0.9.

Table 7.2 summarizes the relevant properties of direct assistance, direct intervention, and decision support. As mentioned earlier, the support model augments the plan library only locally. If no transitions for a particular system action are given for a state, then the execution of the action in this state has no influence on the agent's behavior (the agent then behaves according to the underlying plan library). Note that this does not exclude that the execution of such an action nevertheless is associated with costs (see description of cost-reward-model in subsection 7.2.2).

	Direct Assistance	**Direct Intervention**	**Decision Support**
Goal	support "good" plans	prevent "bad" plans	neutral
Realization	change environment	change environment	provide information
System Role	active	active	passive
Plan Change	no	yes	possibly
Added Value	reduced costs	reduced costs and/or increased rewards	reduced costs and/or increased rewards
Influences	transition function	transition function	action sel. function

Table 7.2: Three classes of support actions can be distinguished.

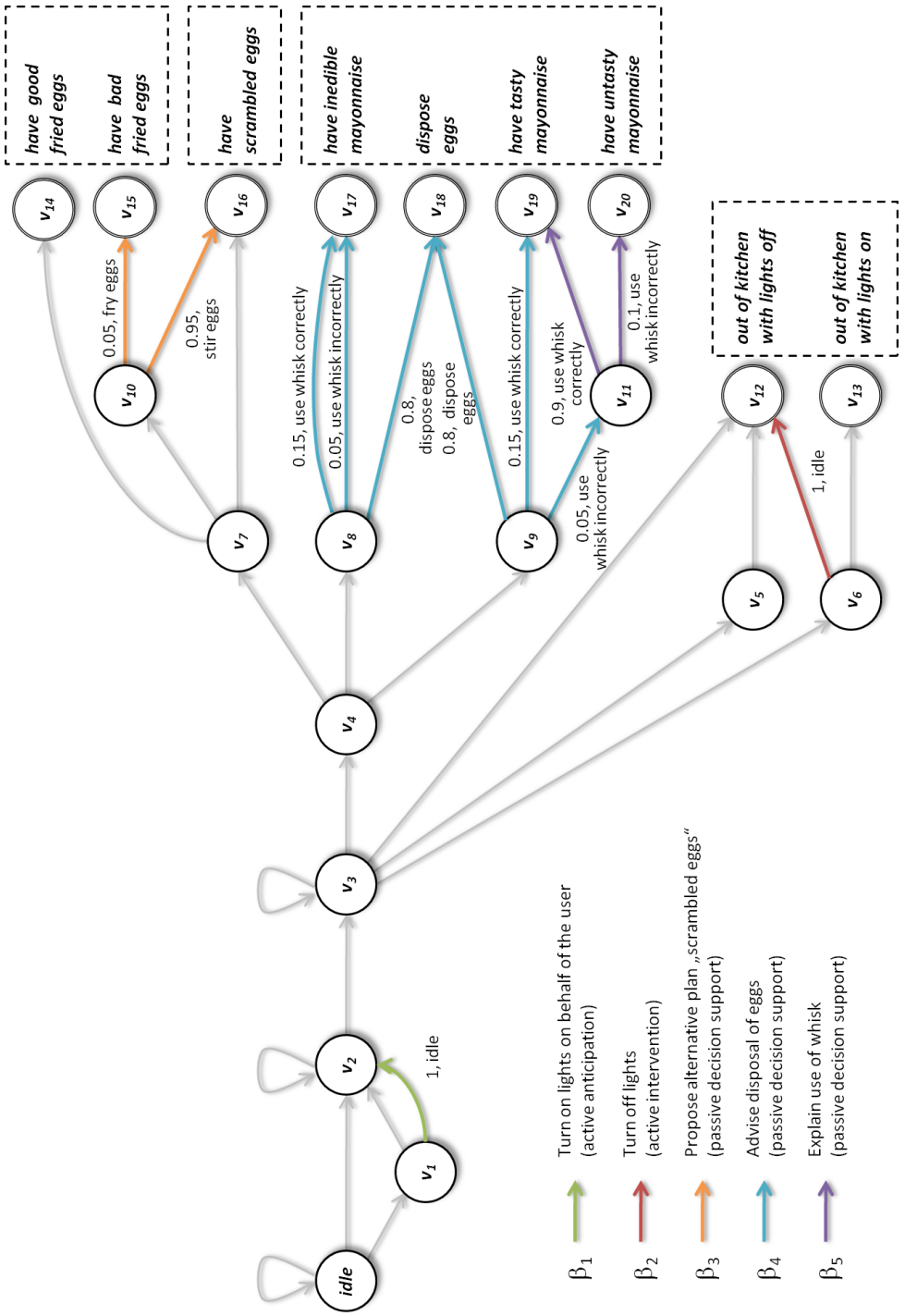


Figure 7.3: The support model locally augments the agent's plan library to describe the effects of system actions.

It is important to note, that whenever the execution of a support action in an "inappropriate" state has side effects which possibly influence the agent's plan selection and execution process, then these side effects have to be represented in the support model as well, and have to be specified individually for each effected system action and state. In our example, β_3 if executed in v_9 has the side effect of incorrectly disposing eggs although they are good.

Formally the support model comprises four elements: A plan library L provides the foundation of the support model as described above, a finite set $\mathcal{B}$ of system actions comprises the system's *support action repertoire*, and two functions s^* (called *support-influenced action selection function*) and t^* (called *support-influenced transition function*) provide a unified view on the local augmentations that are described by the support model. The formal definition of a support model then is as follows:

Definition 7.2.1. A **support model** is a quadruple $(L, \mathcal{B}, s^*, t^*)$, where

- $L = (\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ is a plan library
- $\mathcal{B}$ is a finite set of the system's support actions
- the obligatory action $none \in \mathcal{B}$ represents the case where no support is provided
- $s^* : \mathcal{V} \setminus \mathcal{G} \times \mathcal{B} \times \mathcal{A} \mapsto [0, 1]$ is the support-influenced action selection function, where $s^*(v, \beta, \alpha)$ returns the probability that the agent executes action α as a result of the provision of support action β in state v
- $s^*(v, none, \alpha) = s(v, \alpha)$
- $\forall_{v \in \mathcal{V} \setminus \mathcal{G}, \beta \in \mathcal{B}} : \sum_{\alpha \in \mathcal{A}} s^*(v, \beta, \alpha) = 1$
- $t^* : \mathcal{V} \setminus \mathcal{G} \times \mathcal{B} \times \mathcal{A} \times \mathcal{V} \mapsto [0, 1]$ is the support transition function, where $t^*(v, \beta, \alpha, v')$ returns the probability that the provision of service β during an execution of agent action α in state v results in a transition to state v'
- $t^*(v, none, \alpha, v') = t(v, \alpha, v')$
- $\forall_{v \in \mathcal{V} \setminus \mathcal{G}, \beta \in \mathcal{B}, \alpha \in \mathcal{A}} : s^*(v, \beta, \alpha) > 0 \Rightarrow \sum_{v' \in \mathcal{V}} t^*(v, \beta, \alpha, v') = 1$

The example support model from Figure 7.3 then can be described by a quadruple $(L, \mathcal{B}, s^*, t^*)$ where

- L is the example plan library from subsection 6.2.2
- $\mathcal{B} = \{none, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$
- The values of the support-influenced action selection function s^* are given in Table 7.3
- The values of the support transition function t^* are given in Table 7.4

7.2.2 Cost-Reward Model

In this section we introduce the cost-reward model that describes the costs associated with the execution of agent and system actions and the rewards associated with the achievement of (partial) goals. In the presented model all these factors are subsumed by a single *cost-reward function* which provides the immediate (short-term) outcome utility of a single transition in the plan execution state space. A negative return value of this function represents an excess of costs, while a positive return value represents an excess of rewards.

It may seem difficult to give a function which for each of the manifold combinations of actions and transitions provides a meaningful estimate of costs and rewards. Later in this section we describe how in many practical applications the cost-reward function can be derived from a set of three simpler functions.

Definition 7.2.2. A *cost-reward model* is a quadruple $(\mathcal{V}, \mathcal{A}, \mathcal{B}, r)$, where

- $\mathcal{V}$ is the set of plan execution states that is covered by the cost-reward model
- $\mathcal{A}$ is the set of agent actions that is covered by the cost-reward model
- $\mathcal{B}$ is the set of support actions that is covered by the cost-reward model
- $r : \mathcal{V} \times \mathcal{B} \times \mathcal{A} \times \mathcal{V} \mapsto \mathbb{R}$ is the *cost-reward function*, where $r(v, \beta, \alpha, v')$ returns the system owner's immediate outcome utility of a transition from state v to state v' while system action β and agent action α are executed

State v	**System Action** β	**Agent Action** α	$s^*(v, \beta, \alpha)$
v_1	β_1	idle	1.00
		otherwise	0.00
v_6	β_2	idle	1.00
		otherwise	0.00
v_8	β_4	use whisk correctly	0.15
		use whisk incorrectly	0.05
		dispose eggs	0.80
		otherwise	0.00
v_9	β_4	use whisk correctly	0.15
		use whisk incorrectly	0.05
		dispose eggs	0.80
		otherwise	0.00
v_{10}	β_3	fry eggs	0.05
		stir eggs	0.95
		otherwise	0.00
v_{11}	β_5	use whisk correctly	0.90
		use whisk incorrectly	0.10
		otherwise	0.00
otherwise			$s(v, \alpha)$

Table 7.3: Probability values of example support-influenced action selection function s^*.

Start State v	**Sys. Action** β	**Agent Action** α	**End State** v'	$t^*(v, \beta, \alpha, v')$
v_1	β_1	idle	v_2	1.00
		otherwise		0.00
v_6	β_2	idle	v_{12}	1.00
		otherwise		0.00
otherwise				$t(v, \alpha, v')$

Note: We do not have to redefine the transition probabilities for $\beta_3, \ldots, \beta_5$ in t^* as in the way we modeled these system actions they only influence the user's action *selection* process, which is completely described by function s^* above.

Table 7.4: Probability values of example support transition function t^*.

If a cost-reward model is sufficient to fully describe the costs and rewards of all plans that are defined by a particular plan library, then we say that this support model *completely covers* the plan library.

Definition 7.2.3. A cost-reward model $(\mathcal{V}', \mathcal{A}', \mathcal{B}', u)$ **completely covers** a plan library $(\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ if and only if

- $\mathcal{V} \subseteq \mathcal{V}'$
- $\mathcal{A} \subseteq \mathcal{A}'$

Similarly, we say that a cost-reward model *completely covers* a support model, if the cost-reward model is sufficient to fully describe the costs and rewards that are associated with the execution of all support actions (and the possibly resulting modified state transitions) defined by the support model:

Definition 7.2.4. A cost-reward model $(\mathcal{V}', \mathcal{A}', \mathcal{B}', u)$ **completely covers** a support model $(L, \mathcal{B}, s^*, t^*)$ if and only if

- the support model $(\mathcal{V}', \mathcal{A}', \mathcal{B}', u)$ completely covers L
- $\mathcal{B} \subseteq \mathcal{B}'$

When defining a cost-reward function r it is important to adopt the correct perspective on the encountered costs and rewards. As the purpose of the support system is generally determined by its owner, it is crucial to model costs and rewards from the system owner's point of view. These might or might not correspond to the costs and rewards that result for the observed agent. In the previous chapters we assumed that owner of the system and observed agent are the same entity. In this case, the system owner's costs and rewards equal the agent's costs and rewards. In practical applications it is reasonable to distinguish between the observed agent's personal outcome value and the support system owner's outcome value. The latter one differs from the first one in that it includes the support system owner's interests and the intended purpose of plan-recognition-based support.

As an example imagine that a shop offers a plan-recognition-based service that supports customers in buying products. For the observed customer, the

utility of this service might depend on a combination of increase in satisfaction with the bought products and the amount of money saved due to the service's recommendations. For the shop owner the utility of the same service might depend on a combination of the increase in his profit and the increase in customer satisfaction.

For the definition of cost-reward function r, it is helpful to have a closer look on the relationship between the support system and the observed agent and the resulting interdependencies between agent utility and system utility (respectively the system owner's utility). Similar to plan recognition, which can be performed to recognize plans of friendly or hostile agents (see section 3.1), a support system might either assist a friendly agent in reaching its goals (*assistive support*), hinder a hostile agent from reaching its goals (*adverse support*), or influence an agent's plan execution process independent of the attitude towards it (*neutral support*):

- **Assistive Support** means that the support system has the purpose of supporting the observed agent in executing its plans. This goal is achieved by choosing support actions, which either allow the observed agent to reach the desired goals more efficiently (at a lower cost), and/or help the agent to reach "better" (higher-valued) goals. In the case of assistive support, the added value that is realized through the provision of support correlates with the added value that the observed agent experiences minus the cost of providing this support.

- **Adverse Support** means that the support system has the purpose of disturbing or even preventing the observed agent from executing its plans. This is achieved by choosing support actions which force the agent to invest more resources (generate higher costs) to reach its goal, and/or to make it reach "less desired" (lower-valued) goals. In the case of adverse support, the added value that is realized through the provision of support correlates with the negative added value that the observed agent experiences minus the cost of providing this support.

- **Neutral Support** means that the support system's purpose is neither to explicitly support the observed agent nor to explicitly prevent the observed agent from performing its plans, but to realize some utility which might be (at least partially) independent from the observed agent's outcome utility. In the case of neutral support, the added value that is realized through the provision of support is determined by the individual costs and rewards associated with certain actions and/or goals minus the costs that are associated with the provision of support.

In the cases of assistive and adverse support, one can exploit existing dependencies between system and agent utility to define a *factorized cost-reward model*, which defines the outcome utility by means of a set of three simpler functions. Here we assume that the outcome utility of a transition is comprised by three independent factors: The cost of the agent action, the cost of the support action, and a possible reward for reaching an intermediate or goal state. Each of these factors is represented by an individual function. Furthermore, the factorized model assumes that all three factors are independent of the current state of the agent's plan selection and execution process. The formal definition of the factorized cost-reward model is provided in Definition 7.2.5.

Definition 7.2.5. A **factorized cost-reward model** is a tuple $(\mathcal{V}, \mathcal{A}, \mathcal{B}, c_U, c_S, r_\mathcal{V}, i)$, where

- $\mathcal{V}$ is the set of plan execution states that is covered by the model
- $\mathcal{A}$ is the set of agent actions that is covered by the model
- $\mathcal{B}$ is the set of support actions that is covered by the model
- $c_U : \mathcal{A} \mapsto \mathbb{R}$ is the *agent action cost function*, where $c_U(\alpha)$ returns the cost which the agent has to pay for the execution of α
- $c_S : \mathcal{B} \mapsto \mathbb{R}$ is the *system action cost function*, where $c_S(\beta)$ returns the cost which the system has to pay for the execution of β
- $r_\mathcal{V} : \mathcal{V} \mapsto \mathbb{R}$ is the *state reward function*, where $r_\mathcal{V}(v)$ returns the observed agent's immediate reward of reaching state v
- $i \in \{assistive, adverse\}$ denotes if the model describes the costs and reward of assistive or adverse support

From each factorized cost-reward model $(\mathcal{V}, \mathcal{A}, \mathcal{B}, c_U, c_S, r_\mathcal{V}, i)$ we can always derive a general cost-reward model $(\mathcal{V}', \mathcal{A}', \mathcal{B}', r)$ with

- $\mathcal{V} = \mathcal{V}'$
- $\mathcal{A} = \mathcal{A}'$
- $\mathcal{B} = \mathcal{B}'$
- $r(v, \beta, \alpha, v') = \begin{cases} r_\mathcal{V}(v') - c_U(\alpha) - c_S(\beta) & \text{if } i = assitive \\ -r_\mathcal{V}(v') + c_U(\alpha) - c_S(\beta) & \text{if } i = adverse \end{cases}$

To complete our example from the cooking domain we assume the following cost-reward model: We suppose that the cost of all agent actions except action *idle* equals 5. Agent action *idle* and system action *none* are free of cost, turning on and off the lights (actions β_1 and β_2) are very cheap (cost 1), and all other system actions have a cost of 30 as they require the user to follow and understand some longer advice. Goals are assigned the rewards given in Table 7.5, where for instance the worst outcome is to have (and possibly eat) inedible mayonnaise, and having scrambled eggs is preferred over having bad fried eggs. Based on this assumptions we can formally describe our sample cost-rewards model by the tuple $(\mathcal{V}, \mathcal{A}, \mathcal{B}, c_U, c_S, r_\mathcal{V}, assistive)$ where

- $\mathcal{V}$ equals the state space of the sample plan library we defined in subsection 6.2.2
- $\mathcal{A}$ equals the agent's action repertoire of the sample plan library we defined in subsection 6.2.2
- $\mathcal{B}$ equals the system's set of support actions from the sample support model we defined in subsection 7.2.1.
- $c_U(\alpha) = \begin{cases} 0 & \text{if } \alpha = idle \\ 5 & \text{otherwise} \end{cases}$
- $c_S(\beta) = \begin{cases} 0 & \text{if } \beta = none \\ 1 & \text{if } \beta \in \{\beta_1, \beta_2\} \\ 30 & \text{if } \beta \in \{\beta_3, \beta_4, \beta_5\} \end{cases}$
- The values of the state reward function $r_\mathcal{V}$ are given in Table 7.5

State v	**Description**	$r_\mathcal{V}(v)$
v_{12}	out of kitchen with lights off	0
v_{13}	out of kitchen with lights on	−50
v_{14}	have good fried eggs	200
v_{15}	have bad fried eggs	50
v_{16}	have scrambled eggs	120
v_{17}	have inedible mayonnaise	−5000
v_{18}	dispose eggs	−10
v_{19}	have tasty mayonnaise	200
v_{20}	have untasty mayonnaise	0
otherwise		0

Table 7.5: Values of example state reward function $r_\mathcal{V}$.

7.2.3 POMDP Representation of SDP

In order to compute the best system action and to infer its expected utility for use in Definition 7.1.1, we have to solve decision problem SDP. This section describes how SDP can be solved by using the support and cost-reward models presented above to formulate SDP as a partially observable Markov decision process.

Partially Observable Markov Decision Processes (POMDPs) model decision theoretic planning problems in which an agent must make a sequence of decisions to maximize its utility given uncertainty of the effects of its actions and its current state [CKL94, Whi91]. The model assumes, that at any moment in time the agent is in one of a finite set of possible states S and must choose one of a finite set of possible actions A. After taking action $a \in A$ the agent's state becomes some state $s' \in S$ with a probability given by a transition function p_t. The current state s is hidden to the agent, it can only be indirectly observed by the agent through a finite set of possible (eventually partial) observations O, which originate after the execution of a specific action and a resulting specific transition with a probability given by an observation function p_o. After each transition, the agent receives an immediate reward that depends on the chosen action a, the resulting state transition, and the made observation. This reward is assumed to be given as a reward function r. Defintion 7.2.6 formally defines POMDPs.

Solving a POMDP means finding a policy π that maps a hypothesis on S (represented as a probability distribution over S) to an action $a \in A$, such that the *discounted sum* of rewards is maximized if a is executed. If r_0 is the immediate reward in the current step, r_1 the immediate reward in the next step, and so on, then the discounted sum is defined as

$$\sum_{i=0}^{\infty} q^i r_i$$

where $0 < q < 1$ is the *discount factor* that expresses the system owner's preference for near-term rewards towards long-term rewards. The expected long-term reward for solution a equals the maximum expected utility $\widehat{EU}$ of the sequential decision problem represented by the solved POMDP.

In the case of our support decision problem SDP, the POMDP should model the system's decision on the support action to choose based on possibly incomplete knowledge about the current state of the agent's plan selection and execution process. Accordingly, the POMDP states and transitions represent the agent's plan selection and execution process *from the support system's perspective*. The true plan execution state is not directly accessible to the support system, but hypotheses on the current state are

Definition 7.2.6. A **partially observable Markov decision process** (POMDP) is a tuple (S, A, O, p_t, p_o, r), where

- S is a finite set of states
- A is a finite set of actions
- O is a finite set of observations
- $p_t : S \times A \times S \mapsto [0, 1]$ is the transition probability function, where $p_t(s, a, s')$ returns the probability of a transition to state s' given that action a was executed in state s
- $p_o : A \times S \times O \mapsto [0, 1]$ is the observation probability function, where $p_o(a, s', o)$ returns the probability of observation o given that the execution of action a resulted in a state transition to state s'
- $r : S \times A \times S \times O \mapsto \mathbb{R}$ is the reward function, where $r(s, a, s', o)$ returns the immediate reward which the agent receives given that the execution of action a in state s resulted in a state transition to state s' with the occurrence of observation o

available in form of the plan recognition system's hypotheses. In principle, POMDP states then equal plan execution states, POMDP actions equal system actions, POMDP observations equal agent actions, POMDP transition and observation probabilities are derived from the support model, and the POMDP reward function is derived from the cost-reward model. Next, the SDP POMDP can be solved, and the resulting policy can be used to compute the values of $\widehat{EU}_{SDP}$ and thus EUI_{SDP} as described in Definition 7.1.1.

A problem with this approach is that we cannot establish a one-to-one mapping between plan execution states and POMDP states due to the intended representation of system actions as POMDP actions and agent actions as POMDP observations. In the support model the probability of an agent action depends on the executed system action and *originating* state. In contrast, POMDPs assume that the probability of an observation depends on the chosen action and *destination* state. This subtle but important difference hinders a direct translation of the support model's structure to a POMDP.

This problem can be solved through the introduction of additional helper states in the POMDP representation. For each non-terminal plan execution state v, each system action β, and each agent action α that occurs with a

probability larger than zero in state v given the execution of system action β, a new helper state $v_{\alpha,\beta}$ is added to the set of POMDP states. Intuitively, a helper state $v_{\alpha,\beta}$ explicitly represents the (virtual) state immediately after agent action α was executed in state v as response to system action β, where the final transition in the plan execution state space has not yet occurred. The set of all helper states then is defined as follows:

Definition 7.2.7. The **set of helper states** $\mathcal{V}_H$ of some support model $(L, \mathcal{B}, \mathcal{A}^*, t^*, s^*)$ with plan library $L = (\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ is a finite set of states, where

- $\forall_{v \in \mathcal{V}} \forall_{\alpha \in \mathcal{A}^*} \forall_{\beta \in \mathcal{B}} : s^*(v, \beta, \alpha) > 0 \Rightarrow v_{\alpha,\beta} \in \mathcal{V}_H$
- $\mathcal{V}_H$ contains no other elements

Figure 7.4 demonstrates the introduction of helper states based on our cooking domain example. We study the case that a transition to state v_2 occurs: If we would directly associate the possible observations (which represent executed agent actions) with state v_2, the POMDP formalism would not allow to give their probabilities depending on the originating state. Hence, observation *turn lights on* would be assigned the same probability than observation *enter kitchen*, regardless of whether the agent was assumed to be in state *idle* or state v_1 before. In order to solve this problem we introduce additional states "in the middle" of transitions (blue and green nodes in Figure 7.4[4]) and define the observations for these states instead.

A single transition from state v to v' in the plan execution state space due to some agent action α and system action β then is represented by two transitions in the POMDP: The first transitions represents the action selection step, and occurs from POMDP state v to the helper state $v_{\alpha,\beta}$. The second transition represents the agent's action execution step, and occurs from helper state $v_{\alpha,\beta}$ to POMDP state v'.

We define the reward that is associated with the first transition to be 0, and the reward associated with the second transition to equal the reward of the according plan execution state space transition that is given by the cost-reward model. Accordingly, we assume that observations of executed agent actions occur on the first transition, while on the second transition a

[4]For reasons of clarity Figure 7.4 omits edges and helper states for system actions other than *none* if they equal the ones shown for system action *none*.

default *null* observation occurs (which is omitted from Figure 7.4 for reasons of clarity). Formally, the *support selection POMDP* is defined as follows:

Definition 7.2.8. A **support selection** $POMDP_{SDP}$ of some support model $(L, \mathcal{B}, s^*, t^*)$ with completely covered plan library $L = (\mathcal{V}, \mathcal{G}, \mathcal{A}, s, t)$ and completely covering cost-reward model $(\mathcal{V}', \mathcal{A}', \mathcal{B}', r')$ is a partially observable Markov decision process (S, A, O, p_t, p_o, r), where

- $\mathcal{V}_H$ is the set of helper states for support model $(L, \mathcal{B}, s^*, t^*)$ and plan library L
- $S = \mathcal{V} \cup \mathcal{V}_H$
- $A = \mathcal{B}$
- $O = \mathcal{A} \cup \{null\}$
- $p_t(s, a, s') = \begin{cases} s^*(v, \beta, \alpha) & \text{if } s = v \in \mathcal{V} \wedge s' = v'_{\alpha,\beta} \in \mathcal{V}_H \\ t^*(v, \beta, \alpha, v') & \text{if } s = v_{\alpha,\beta} \in \mathcal{V}_H \wedge s' = v' \in \mathcal{V} \\ 0 & \text{otherwise} \end{cases}$
- $p_o(a, s', o) = \begin{cases} 1 & \text{if } s' = v' \in \mathcal{V} \wedge o = null \\ 1 & \text{if } s' = v_{\alpha,\beta} \in \mathcal{V}_H \wedge o = \alpha \\ 0 & \text{otherwise} \end{cases}$
- $r(s, a, s', o) = \begin{cases} r'(v, \beta, \alpha, v') & \text{if } s = v_{\alpha,\beta} \in \mathcal{V}_H \wedge s' = v' \in \mathcal{V} \\ 0 & \text{otherwise} \end{cases}$

The constructed $POMDP_{SDP}$ can now be solved offline with approaches like [CLZ97],[PGT03], or [SS05]. The resulting policy then can be used to calculate $\widehat{EU}_{SDP}$ and EUS_s according to Definition 7.1.1.

In the following we demonstrate these calculations in our cooking domain example. After the introduction of helper states as described above the resulting POMDP comprises 165 states[5]. We solve this POMDP with ZMDP [Smi09], an implementation based on [SHDC06] in under 1 second on an Intel Core2 Duo 6600 CPU with 2.4 GHz. For the case of complete

[5]The resulting POMDP can be compacted to 54 states by merging helper states in parts of the plan library that are not augmented by the support model.

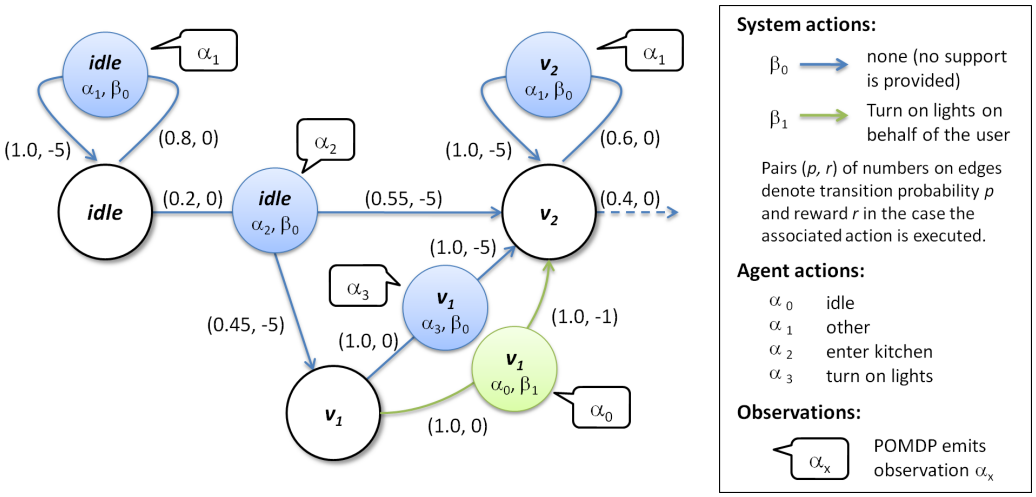


Figure 7.4: Helper states (blue and green nodes) allow for the correct representation of observation probabilities in the POMDP.

information (which means that we exactly know in which state the observed agent currently is) and a discount factor of 0.95 the resulting policy returns for each non-goal state the optimal system action and associated maximum expected utility values shown in Table 7.6.

From the computed policy we now can derive the optimal system action and associated maximum expected utility for any possible hypothesis. For instance if for some reason we come to the conclusion that the current hypothesis is $h_t = \{v_8 : 0.11, v_9 : 0.89\}$ then the policy allows us to compute that the maximum expected utility of $\widehat{EU}_{SDP}(h_t) = -114.45$ is reached for system action β_4. Hence our support system should decide to execute this action.

Next we use the inferred policy together with the equation from Definition 7.1.1 to compute the expected utility of observation source information for the sensors in our cooking domain sample models. For this example we compute for every non-goal state the utility values for all sensors and pick the one with the highest utility rating. Table 7.7 summarizes the results: Each row gives the sensor with the highest expected utility for a particular state. We can see that our utility function "proposes" suitable sensors to recognize all existing support opportunities.

A dash in a table row denotes the case where none of the available sensor has an expected utility of larger than 0. This happens when no (single) sensor allows to reduce the uncertainty to an extend which changes the support systems behavior (and thus generates added value). For this case, two

Hypothesis h	**Best System Action** β	$\widehat{EU}_{SDP}(h)$
$h = \{idle : 1.0\}$	*none*	−9.38
$h = \{v_1 : 1.0\}$	β_1 (turn on lights)	11.16
$h = \{v_2 : 1.0\}$	*none*	13.41
$h = \{v_3 : 1.0\}$	*none*	30.20
$h = \{v_4 : 1.0\}$	*none*	60.20
$h = \{v_5 : 1.0\}$	*none*	−4.75
$h = \{v_6 : 1.0\}$	β_2 (turn off lights)	−0.95
$h = \{v_7 : 1.0\}$	*none*	147.64
$h = \{v_8 : 1.0\}$	β_4 (advise disposal of eggs)	−990.85
$h = \{v_9 : 1.0\}$	*none*	145.25
$h = \{v_{10} : 1.0\}$	β_3 (propose scrambled eggs)	77.43
$h = \{v_{11} : 1.0\}$	β_5 (explain use of whisk)	137.75

Table 7.6: Best system actions and associated maximum expected utility $\widehat{EU}_{SDP}$ (rounded) in our cooking domain example (case of complete information).

possible reasons exist: One the one hand, the sensor set simply might not contain a suitable sensor, although a sensor would be helpful to improve the recognition of the plan. On the other hand, sometimes not every step of a plan needs to be recognized, for instance if no support can or should be provided during certain parts of the plan, or because a plan contains a unique observation which alone is sufficient to identify the plan.

In order to find out which of the above mentioned cases holds, one can use a simulation approach like the one we present in our evaluation (see chapter 10). For a known plan library, such a simulation can be performed once with the original sensor model and once with an extended sensor model that contains additional sensors. By comparing the recognition performance of both, one can judge which set of sensors is better suited to support the plan recognition process. With the same approach, one can also determine the optimal position for new sensors by creating and evaluating different sensor models for each sensor arrangement.

7.3 Refining Plan Recognition

The support system's influence on the agent's plan selection and execution process is not only important to be considered in the context of SDP and

Hypothesis h	**Best Sensor** s	**Expected Utility** $EUS_s(h)$
$h = \{idle : 1.0\}$	*cam door*	0.25
$h = \{v_1 : 1.0\}$	-	0.00
$h = \{v_2 : 1.0\}$	-	0.00
$h = \{v_3 : 1.0\}$	*cam door*	0.54
$h = \{v_4 : 1.0\}$	*rfid counter*	38.05
$h = \{v_5 : 1.0\}$	-	0.00
$h = \{v_6 : 1.0\}$	-	0.00
$h = \{v_7 : 1.0\}$	*cam stove*	1.98
$h = \{v_8 : 1.0\}$	-	0.00
$h = \{v_9 : 1.0\}$	*wsn whisk*	27.50
$h = \{v_{10} : 1.0\}$	-	0.00
$h = \{v_{11} : 1.0\}$	-	0.00

Table 7.7: Best-ranked sensors with expected utility value for every possible plan execution state in our example (rounded, case of complete information).

the assessment of observation information utility, but also in the basic plan recognition process itself. The state-based plan recognition approach that was presented in chapter 3 and applied so far assumed that the evolution of a plan (and thus the agent's behavior) depends on the current plan execution state in a way that is completely described by the plan library. With the introduction of system actions, this assumption is no longer valid. Now, the agent's behavior additionally depends on the executed support action in a way that is described by the support model.

In order to account for this new situation, the combined plan recognition DBN introduced in section 6.4 has to be adapted in order to additionally consider the influence of support actions on the agent's plan selection and execution process. This need is addressed by augmenting the original network with an additional *System Action* node with domain $\mathcal{B}$, which represents the support action that is executed by the system. Figure 7.5 shows the new structure of the resulting *extended plan recognition DBN*. Via the introduction of two new edges, the value of this node influences the value of the *Agent Action* node and the *New State* node, as already described in the presentation of the support model (see subsection 7.2.1). The conditional probability tables of both nodes are given by the support model's functions s^* and t^*.

The new structure of the plan recognition DBN requires minor changes to existing models and algorithms: The plan recognition algorithm presented in section 6.4 has to be provided with the executed support action as additional

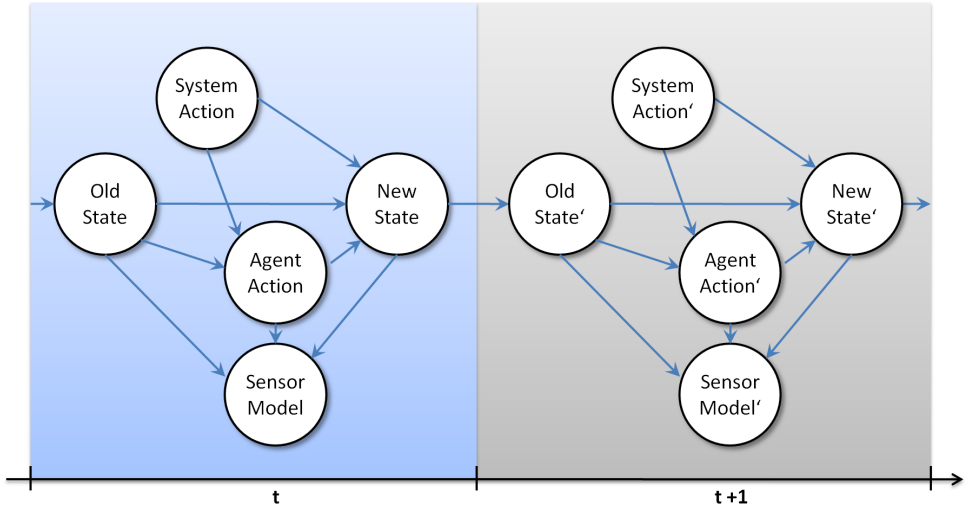


Figure 7.5: The *extended plan recognition DBN* considers the influence of support actions on the agent's plan selection and execution process.

prior evidence for the *System Action* node in the extended plan recognition DBN. As this action is chosen by the system itself, this information is always fully known. In a strict formal sense, we would have to extend the plan recognition function's signature by the executed system action as a third argument. As a consequence, all other functions that make use of the plan recognition function would have to be extended too. Although formally required, such an extension is not essential for the further understanding (with one exception discussed below), and thus is omitted for reasons of clarity. In the following we simply assume that the plan recognition algorithm knows the last executed support action and considers it as required.

The only exception to this rule is the prediction of goals and future actions, as described in section 6.5. Here, the plan recognition algorithm has no chance of "knowing" the last support action, as the according decisions have not yet been made by the higher-order support system. Here, two possible solutions exist: The first option is to ignore the support system and always assume that the support system executes the special-purpose *none* action in future time slices. This leads to prediction results which equal the original case considered in section 6.5. The second alternative is to solve SDP after each prediction step, and then to assume the resulting support action as executed system action in the next iteration of prediction.

Both approaches have their specific pros and cons: Assuming the *absence of any system support* (respectively the execution of *none* actions through

the system) on the one hand is *computationally less expensive*, and on the other hand allows for *predicting the agent's original, uninfluenced plans* and actions. Assuming the *presence of system support* (respectively the execution of support actions which in each iteration of prediction provide the solution to SDP) increases the accuracy of predictions regarding the *finally executed plans*, but on the downside is *computationally more expensive*. For the implementation of sensor selection no prediction about more than one iteration into the future is required, thus this differences can be safely ignored.

7.4 Decision-Theoretic Sensor Selection

The main motivation for the development of the utility model that we present in this chapter is the problem of sensor selection (see section 2.4) in plan recognition applications. In Definition 7.1.1 we already gave a function that allows to compute the expected utility of a single observation sources for a given hypothesis. In order to apply this utility function in sensor selection, we first have to extend it from single sensors to sets of multiple sensors as input. We can then use the resulting function to perform utility-based sensor selection according to Definitions 2.4.1 and 2.4.2.

After giving the extended observation source utility function we describe two exemplary sensor selection strategies DT-BNB and DT-GREEDY, which make use of function EUS to decide on the best subset of sensors to activate. The performance of both strategies is evaluated in chapter 10.

7.4.1 Expected Utility of Sets of Information Sources

Recall from Definition 7.1.1 that function EUS_s (among other variables) computes the weighted sum of utilities over all possible readings. If we want to compute the expected utility of a set of sensors, we have to compute the weighted sum over the crossproduct of all individual sensor's readings. The probability of a reading from the resulting crossproduct then is given as the product of the individual readings probabilities (when constructing our sensor model we assumed conditional independence between sensors (see section 6.3). With this modifications we can then apply the equation from Definition 7.1.1 as usual.

7.4.2 Sensor Selection Strategy DT-BNB

Sensor selection strategy DT-BNB uses a branch and bound approach to solve the utility-based sensor selection problem (see Definition 2.4.2). *Branch and*

Bound (BnB) [LW66] is a general algorithm for finding optimal solutions of various optimization problems. The algorithm systematically enumerates candidate solutions in a search tree. The search is optimized by discarding fruitless subtrees as early as possible by using upper and lower bounds of the quantity being optimized.

Algorithm 2 shows the general recursive branch and bound algorithm that maximizes the target value. Two application-dependent functions adapt it to the specific use case: A *branching* function "branch" partitions the search space (performs one iteration of search tree expansion), while a *bounding* function "bound" computes upper and lower bounds of a partial search space (subtree). In the following set and tree notions are used synonymously.

Algorithm 2: General *Branch and Bound* maximization algorithm

```
branch-and-bound (T, m):
Data: T: Set of candidate solutions; m: Lower bound (initially −∞)
Result: (t̂, m̂)  t̂: Best solution of T; m̂: Lower bound of t̂
begin
    if |T| = 1 then
        return (T,bound (T))
    else
        T′ =branch (T)
        t̂ = ∅
        m̂ = m
        for t ∈ T′ do
            b =bound (t)
            if b > m̂ then
                (t′, m′) =branch-and-bound (t, m̂)
                if m′ > m̂ then
                    t̂ = t′
                    m̂ = m′
                end
            end
        end
        return (t̂, m̂)
    end
end
```

Function *bound* is expected to provide an upper bound on the value of a set of candidate solutions. For a singleton candidate set it is expected to return the exact value associated with the contained solution, which at the

same time is upper and also lower bound of this solution. The algorithm then works as follows: The algorithm maintains a global lower bound which is determined by the value of the best solution found so far. It uses function *branch* to divide the current set of candidate solutions into two or more disjoint subsets, and uses function *bound* to compute the upper bound of each subset. A subset can be safely discarded if its upper bound is below the lower bound of the best solution found so far. Otherwise, the resulting set is recursively expanded until a singleton candidate solution is reached or all of its own subsets have been discarded.

In the case of sensor selection, candidate solutions correspond to sets of selected sensors. The quantity being optimized is the expected utility of information sources, which is computed from the proposed utility model as described in the previous section. The resulting search tree for a set $S = \{s_1, \ldots, s_n\}$ of sensors can be visualized as shown in Figure 7.6 if we represent the progress of our search as a list of length n, where the i^{th} element is '1' if sensor s_i is selected, '0' if sensor s_i is *not* selected, and s_i if no decision about sensor s_i has been made yet.

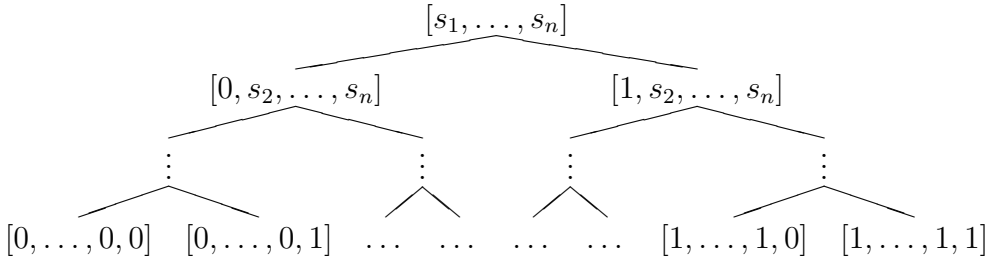


Figure 7.6: Sensor selection search tree for sensors $\{s_1, \ldots, s_n\}$. At tree level i sensor s_i is selected (represented by '1') or unselected (represented by '0').

Algorithm 3 gives the concrete functions *branch* and *bound* for the case of utility-based sensor selection, and shows how to invoke the branch and bound algorithm in the case of sensor selection strategy DT-BNB. In this case, solutions are sets of sensors, hence the set of all candidate solutions is the power set 2^S of all sensors.

Function *branch* partitions the current set T of candidate solutions into two disjoint subsets T_0 and T_1. On recursion level i the first set comprises all candidate sensor sets from T that *do not* contain sensor s_i, while the latter set comprises all candidate sensor sets that contain sensor s_i. Note that this implies that after iteration i all members of a resulting family of sensor sets T contain the same subset of sensors $\{s_1, \ldots, s_i\}$, or formally

$$(\cup_{t \in T} t) \cap \{s_1, \ldots, s_i\} = \cap_{t \in T} t$$

Function *bound* calculates the upper bound of a set T of candidate solutions as follows: If all candidate solutions in T exceed the allowed resource limit, then the whole candidate set is invalid and $-\infty$ is returned as upper bound, which leads to the immediate rejection of the considered candidate set by the main *branch-and-bound* function. If at least one valid solution exists in the candidate set, then we can use the expected sensor utility function EUS_S as described in subsection 7.4.1 to calculate the upper bound of the candidate set.

For non-singleton candidate sets we exploit the decision-theoretic property that additional information never has a negative impact on the resulting outcome utility. This property is due to the fact that a decision maker always has the choice of ignoring additional information if it does not help the decision making. With this property, the upper bound for a set of candidate solutions T is bound by the expected utility of the maximum candidate solution in T. In our search tree notion (see above) the maximum candidate solution in iteration i has the general form $[s_1, \ldots, s_i, 1, \ldots, 1]$, where the values of the first i elements depend on the concrete subtree that is considered, and in particular are constant for all elements of a given set of candidate solutions T (see above). The maximum candidate solution comprises the subset of sensors $\{s_1, \ldots, s_i\}$ that has been selected in earlier branch steps merged with the complete set $\{s_{i+1}, \ldots, s_n\}$ of sensors that we have not decided on yet.

As one can easily see from Algorithm 3, the design of the proposed *branch* step ensures that the desired maximum solution which we use to calculate the upper bound for a given candidate set is always part of that candidate set, and in particular equals the largest sensor set in any candidate set T. As such, the maximum candidate solution always is uniquely defined. We now can easily apply the proposed utility function for observations sources u on the maximum candidate solution to calculate the upper bound of the whole candidate set. As a desired side effect, this approach provides us with the exact value of a solution in the special case of a singleton candidate set, which at the same time provides a lower bound as required by the general branch and bound algorithm.

7.4.3 Sensor Selection Strategy DT-GREEDY

Although sensor selection strategy DT-BNB tries to optimize the search process by discarding fruitless branches of the search tree, the strategy in the

Algorithm 3: Optimal sensor selection strategy DT-BNB

select $_{DT-BNB}(S, \hat{r})$:
Data: $S = \{s_1, \ldots, s_n\}$: Set of all sensors; $\hat{r}$: Upper resource limit
Result: Optimal set of sensors to activate
begin
  $(\{S'\}, \_)$ =**branch-and-bound** $(2^S, -\infty)$
  return S'
end

branch (T):
Data: T: Set of candidate solutions (family of sensor sets)
Result: Set of two disjoint sets of candidate solutions
begin
  for $i = 1, \ldots, n$ **do**
    $T_0 = \{t \in T : s_i \notin t\}$
    $T_1 = \{t \in T : s_i \in t\}$
    if $|T_0| = |T_1|$ **then**
      return $\{T_0, T_1\}$
    end
  end
end

bound (T):
Data: T: Set of candidate solutions (family of sensors sets)
Result: Upper bound on value of best solution in T
begin
  if $\forall_{t \in T} : \sum_{s \in t} r(s) > \hat{r}$ **then**
    return $-\infty$
  else
    return $u(\arg\max_{t \in T} |t|)$
  end
end

worst case might end up with expanding the whole search tree respectively enumerating all possible solutions. In the case of n sensors, there are 2^n possible subsets, which makes full enumeration obviously impractical unless n is very small. As a general drawback one can not tell in advance how many candidate solutions a branch and bound algorithm has to enumerate.

In this section, we thus present a heuristic sensor selection strategy DT-GREEDY, which uses a naïve greedy approach to find a subset of sensors which approximately solves the sensor selection problem.

The strategy starts with an empty set of selected sensors and iteratively adds sensors to this set until the given resource limit is exhausted. In each iteration the strategy computes the expected utility of adding each unselected sensor to the set of already selected sensors. It then adds the sensor which yields the maximum combined expected utility. Algorithm 4 describes the exact behavior in pseudo code.

Algorithm 4: Approximate sensor selection strategy DT-GREEDY

select $_{DT-GREEDY}(S, \hat{r})$:
Data: S: Set of all sensors; $\hat{r}$: Upper resource limit
Result: Optimal set of sensors to activate
begin
  $S' = \emptyset$
  $r' = 0$
  $C = \{s \in S : r(s) \leq \hat{r}\}$
  while $C \neq \emptyset$ **do**
    $s' = \arg\max_{c \in C} u(S' \cup \{c\})$
    $S' = S' \cup \{s'\}$
    $r' = r' + r(s')$
    $C = \{s \in S \setminus S' : r(s) \leq \hat{r} - r'\}$
  end
  return S'
end

The presented sensor selection strategies are just two examples of possible utility-based sensor selection strategies. Other strategies might be based on approaches such as evolutionary algorithms [YSK93], randomized rounding [MR95], or any other exact or approximative method to solve optimizations problems of the form given in Definition 2.4.2.

7.5 Summary

In this chapter we introduced a decision-theoretic utility model for observation information in plan recognition applications that can be used with general utility-based sensor selection approaches to solve the problem of sensor selection in plan recognition applications. This model is based on the state-aware plan recognition approach presented in chapter 6. At first we motivated why it is reasonable to define observation information utility based on the expected utility of the *support decision problem*. This problem is for an external decision-maker component to choose the best support action based on the plan recognition system's hypothesis. Next we explained how to solve the support decision problem by formulating it as a Partially Observable Markov Decision Process (POMDP), and how to use its solution to compute the *expected utility of observation information* in plan recognition applications. Based on this utility measure we described two sensor selection strategies which solve the problem of sensor selection in plan recognition applications. The theoretical models and algorithms proposed in this chapter have been implemented in our RePReTo system that we describe in chapter 8. The performance of the proposed utility model and sensor selection algorithms is evaluated in chapter 10.

8

RePReTo – A Resource-Aware Plan Recognition Tool Set for Instrumented Environments

The previous chapters laid the theoretical foundation for state-aware plan recognition in instrumented environments and developed a utility model for observation information in plan recognition applications. This chapter considers aspects related to the practical application of the proposed theoretical models and methods in real-world application scenarios.

The chapter starts with a description of the RePReTo system, a Java-based library that provides plan recognition, sensor selection, and decision support capabilities to existing and future applications. RePReTo is an acronym for *Re*source-aware *P*lan *Re*cognition *To*ol Set. We present the central components (see section 8.1) and discuss the temporal interaction of these components with each other and their synchronization with the activities in the environment (see section 8.2). Section 8.3 gives a short excursus on the acquisition of required knowledge models in real-world application domains. Section 8.4 concludes this chapter with the presentation of design patterns for state-based plan libraries, which can be applied to ease the manual development or automated composition of plan libraries.

8.1 RePReTo Architecture

Figure 8.1 provides an overview of all components which together comprise the RePReTo system. Rectangular boxes with a straight bottom edge denote software components which perform some kind of computation. Boxes with a waved bottom edge denote knowledge and data models, which are evaluated and used by software components (for reasons of simplicity, models will also be called "components" in the following).

Directed arcs illustrate the flow of information between components. The component the arc is pointing to uses (or is influenced by) information which is provided by the component which resides at the origin of the arc. Arcs in Figure 8.1 do not imply any specific temporal ordering of component interactions (the same holds for the derived Figures 8.2 to 8.4 in the following chapters). Temporal dependencies between reasoning processes are an important aspect of the overall architecture and are discussed in section 8.2.

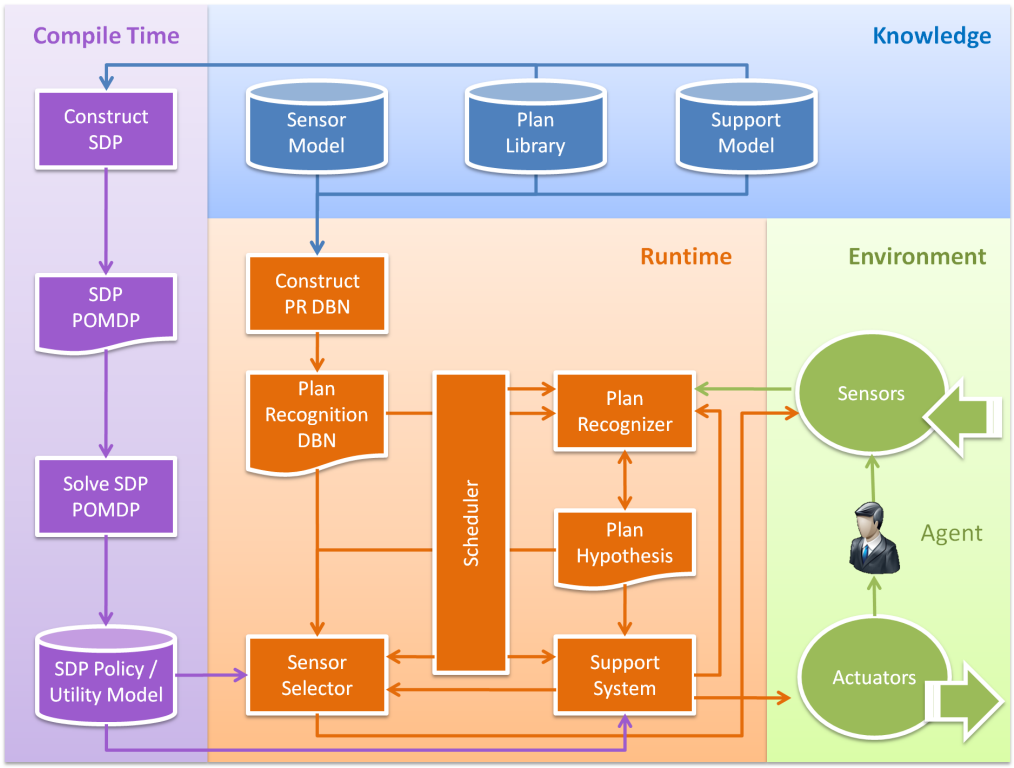


Figure 8.1: Components of the RePReTo architecture. Software components (straight bottom box) and knowledge models (waved bottom box) are connected by directed arcs which illustrate the flow of information.

All components of the RePReTo system can be attributed to one of four layers (in Figure 8.1 layers are distinguished by their color):

- **Knowledge Layer:** This layer comprises the probabilistic models which describe the observed agent's behavior (plan library), the influence of the system's support actions on the agent's behavior (support model), and the relationship between plan execution, support actions, and the resulting sensor observations (sensor model). It is assumed

that these models already exist and are given prior to any computation (see section 8.3 for a discussion on how to create these models). Figure 8.1 shows the components of the knowledge layer in blue color.

- **Compile Time Layer:** This layer comprises all components which perform initial tasks that only have to be performed once in a while, but might require extensive computations which are too costly to perform at runtime. This mainly concerns the generation and solution of the support decision problem, which is computationally hard (see subsection 8.1.2) but only has to be (re)computed in case changes are made to the plan library and/or support model. Figure 8.1 shows the components of the compile time layer in purple color.

- **Runtime Layer:** This layer comprises all components which perform the system's reasoning at runtime. This includes performing the actual plan recognition process, the resource-aware selection of a suitable subset of sensors, and the selection and execution of support functions. For the required reasoning, these components utilize knowledge available from the knowledge layer and the *SDP policy*, which was precomputed during compile time (see subsection 8.1.2). Figure 8.1 shows the components of the runtime layer in red color.

- **Environment Layer:** This layer comprise the components which realize the interface between the physical environment and the considered software system. This includes the set of sensors which are used to observe the agent's behavior. It is assumed that some software interface exists which allows for the activation and deactivation of individual sensors through the system, and that only activated sensors provide observation data to the system. In addition, the environment layer includes a set of actuators, which allow the support system to influence the observed agent's plan selection and execution process. Possible actuators include multi-modal information displays, remotely-operated devices, or any other service which allows to influence the agent's or environment's state. Figure 8.1 shows the components of the environment layer in green color.

In the following the involved components and their interdependencies are described in more detail. To allow for a more structured presentation, the components will be introduced in three groups: First, all components which are involved in realizing the core plan recognition functionality are explained. In a second step, the components which realize the support functionality of the overall system are introduced. Finally, the components required for sensor selection are discussed.

8.1.1 Plan Recognition Components

RePReTo's core plan recognition functionality is realized by the components highlighted in Figure 8.2. A *plan library* (see section 6.2) and a compatible *sensor model* (see section 6.3) provide a statistical description of the agent's plan selection and execution process respectively the observations that might be made by the environment's sensors as a result of the executed plan. Both models are used to construct the dynamic Bayesian plan recognition network (*plan recognition DBN*, see section 6.4). The construction process is cheap and can easily be performed at runtime. This is useful to account for dynamic changes in the sensor model. For the moment, the influence of performed support actions on the agent's plan execution process should be ignored (this aspect is discussed in subsection 8.1.2).

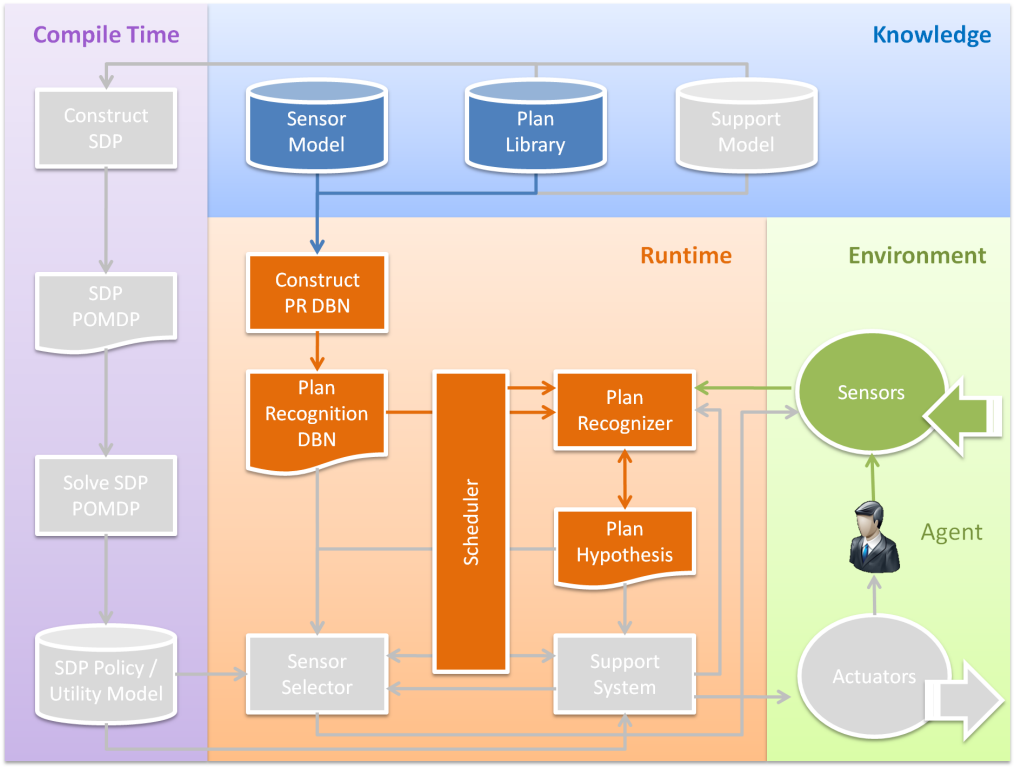


Figure 8.2: Components of the RePReTo system which realize the core plan recognition functionality (highlighted).

The *plan recognition DBN* is used by the *plan recognizer* component to interpret the observations which are provided by the set of active *sensors*. The *plan recognizer* feeds these observations as evidence into the *plan recog-*

nition DBN to incrementally update the current *plan hypothesis* according to the plan recognition algorithm that is described in section 6.4. The *plan recognizer* component is invoked by the *scheduler* component, which centrally coordinates the invocation of components and thus ensures their correct temporal interplay in synchronization with the agent's plan selection and execution process (see section 8.2).

The updated *plan hypothesis* provides a compact representation of the system's current belief about the state of the agent's plan selection and execution process. As it will be shown in the following sections, this hypothesis provides the foundation of all other system functions like the provision of support or the resource-aware selection of sensors.

8.1.2 Support Components

RePReTo's support functionality is realized by the components highlighted in Figure 8.3. The main task of these components is to solve the support decision problem SDP. This requires a statistical model of how and at which costs the system's actions influence the agent's plan execution process, the so-called *support model* (see subsection 7.2.1). This model is used together with the *plan library* to *construct the support decision problem* by formulating it as a *SDP POMDP* (see subsection 7.2.3). Next we *solve the SDP POMDP*, which means to find a *policy* that for each plan hypothesis allows to infer the best support action and its expected utility. Solving a POMDP is a standard problem for which existing algorithms like exact incremental pruning [CLZ97] or anytime point-based value iteration [PGT03] can be used.

Depending on the size of the *plan library* and *support model* (and thus the size of the resulting *SDP POMDP*), the process of solving the POMDP can be very time consuming and thus generally cannot be performed at runtime. However, the POMDP can be constructed and solved offline in advance, if the resulting *SDP policy* is stored for later use. As long as *plan library* and *support model* do not change, *SDP POMDP* does not have to be reconstructed and solved again.

At runtime, the precomputed *SDP policy* can be cheaply evaluated in order to identify the best support action given the current *plan hypothesis*[1].

[1]For a POMDP with n states a policy consists of a finite set of hyper-planes in the belief space $\mathbb{R}^{n-1}$, were each plane is labeled with an action. Each plane is expressed as a vector (its equation coefficients) and likewise each belief state (the probability at each state) is expressed as a vector. The optimal plane is the plane which has the largest dot product with the belief state (which at the same time is the expected utility), and the label of this plane is the optimal action to perform. Hence, the complexity of evaluating a policy is linear in the number of hyper-planes.

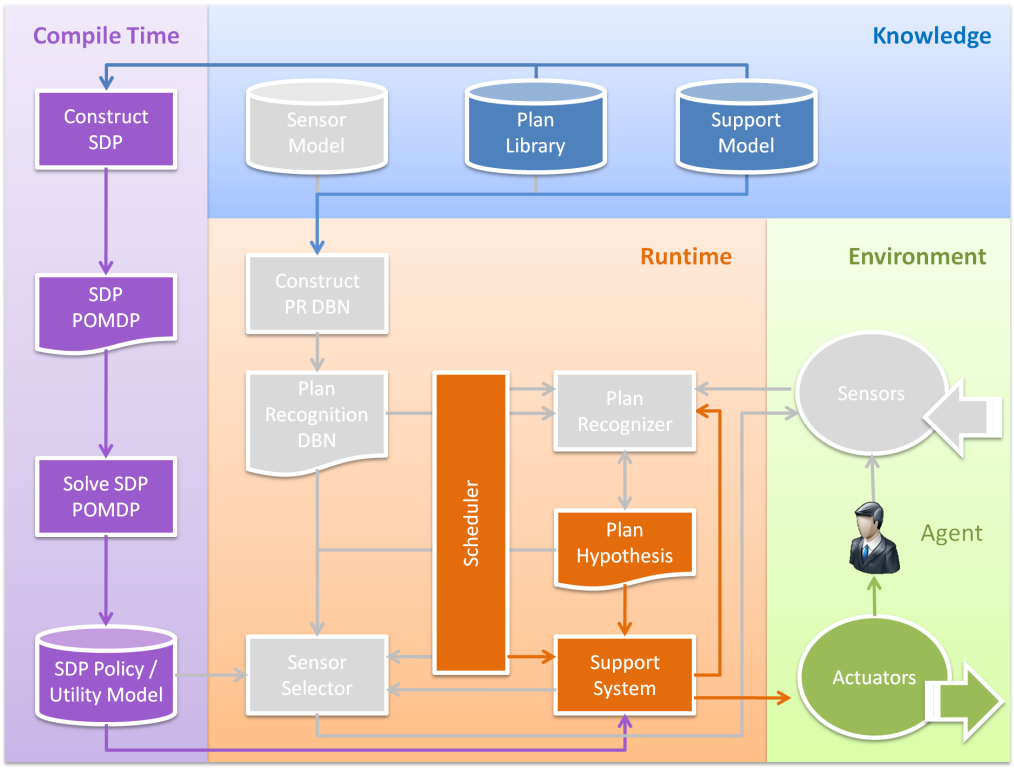


Figure 8.3: Components of the REPRETO system which realize the system's support functionality (highlighted).

This computation is performed by the *support system* component, which then triggers the execution of the selected action through the environment's *actuators*, which in turn influences the agent's plan selection and execution process. Again, this process is controlled by the *scheduler* to ensure the correct temporal behavior of the overall system (see section 8.2).

Note that the execution of support actions possibly influences the agent's behavior and thus needs to be taken into account when performing *plan recognition* (see section 7.3). In Figure 8.3 this relationship is expressed by the additional arcs from the *support model* to the *plan recognition DBN* and from the *support system* to the *plan recognizer* component.

8.1.3 Sensor Selection Components

The components highlighted in Figure 8.4 realize REPRETO's sensor selection functionality. The *sensor selector* component implements the sensor selection algorithm, which chooses the best subset of sensors given the current

plan hypothesis and selected support action. For this purpose, the expected utility of observation information is judged by the utility model for sensor information that is defined in chapter 7. REPRETO does not depend on a particular sensor selection strategy, but can utilize any (exact or approximate) utility-based strategy.

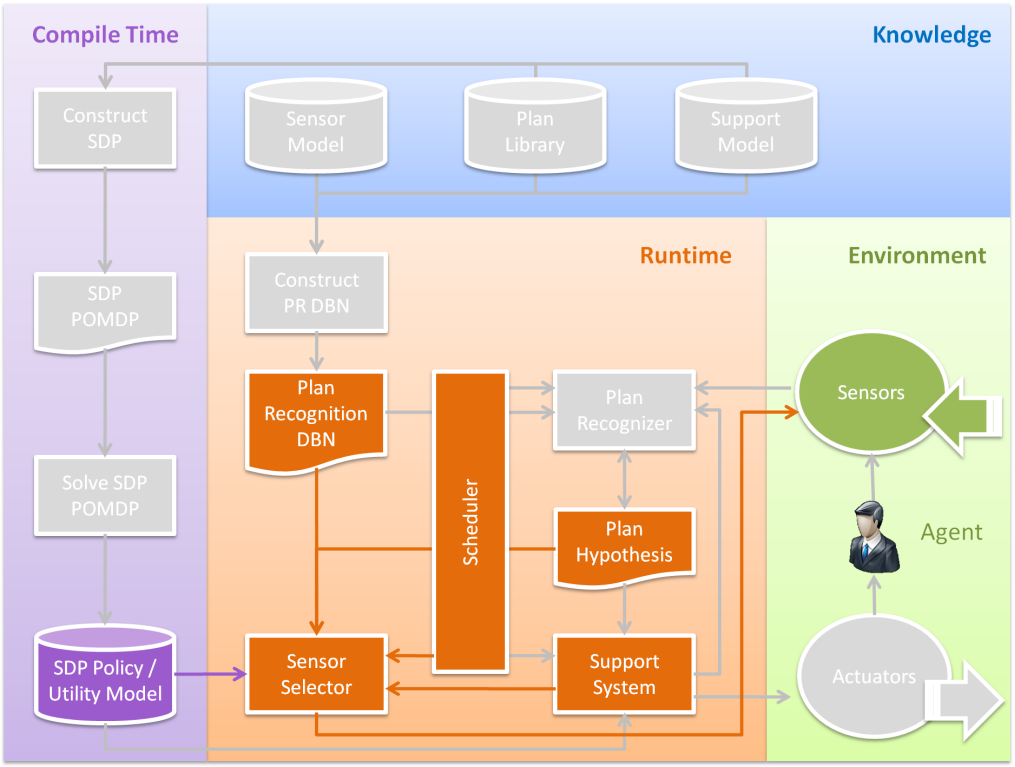


Figure 8.4: Components of the REPRETO system which realize the sensor selection functionality (highlighted).

During the sensor selection process, the *sensor selector* can use the *plan recognition DBN* to predict possible candidate observations for different subsets of sensors together with their a-priori probabilities. The *plan recognition DBN* might furthermore be used by the *sensor selector* to predict the expected influence of candidate observations on the expected plan hypothesis in the next iteration of the plan recognition process.

For each resulting expected plan hypothesis, the *sensor selector* then uses the precomputed *SDP policy* to solve the support decision problem *SDP*. The added value realized by the best possible support action then is used to judge the value of observation information and thus to judge the expected

value of the underlying subset of sensors (see Definition 7.1.1. Whenever predictions about observations, hypothesis, and costs have to be made, the current *plan hypothesis* as well as the current support action that has been selected by the *support system* have to be taken into account.

Finally, the *sensor selector* decides on the most promising (with respect to expected utility of information) subset of sensors given an upper resource limit, and finally activates the chosen subset of *sensors*. As in the case of the *plan recognizer* and *support system* components, the *sensor selector* component is invoked by the central *scheduler* component (see section 8.2).

8.2 Temporal Dependencies

The previous sections of this chapter discussed the logical dependencies between the components of the RePReTo system with respect to the flow of information. This section focuses on the temporal synchronization of the system's components among each other in general, and with respect to the observed agent's plan selection and execution process in particular. As it is discussed below, the correct temporal orchestration of all components is important for the successful recognition of the observed agent's plans, as well as for the subsequent provision of proactive support. To ensure the correct overall timing, a central *scheduler* component controls the invocation of the necessary components and routines.

8.2.1 Turn-Based Scheduling of System Components

In the proposed framework the theoretical foundation for plan recognition is the state-based plan selection and execution model of the observed agent (see section 6.2). This model assumes that the agent chooses and performs his action in discrete turns (in the following denoted as iterations). In order to use this model for plan recognition, the plan recognition process has to mirror this turn-based model in its own reasoning. The general idea is, that one iteration of the agent's plan selection and execution process matches one iteration of the system's plan recognition and support provision process. In the following, the most central temporal dependencies between the system's subprocesses and the agent's behavior during one iteration of plan execution/plan recognition are explained.

A graphical representation of the existing temporal dependencies is shown in Figure 8.5. White-bordered boxes represent software components of the RePReTo architecture. Black-bordered boxes represent concrete procedures that are executed by the system or the observed agent. The temporal

order is given by the spatial position of the boxes within the diagram: Time progresses vertically from top to bottom (primarily), and horizontally from left to right (secondarily). Arcs represent the sequence of component and procedure invocations.

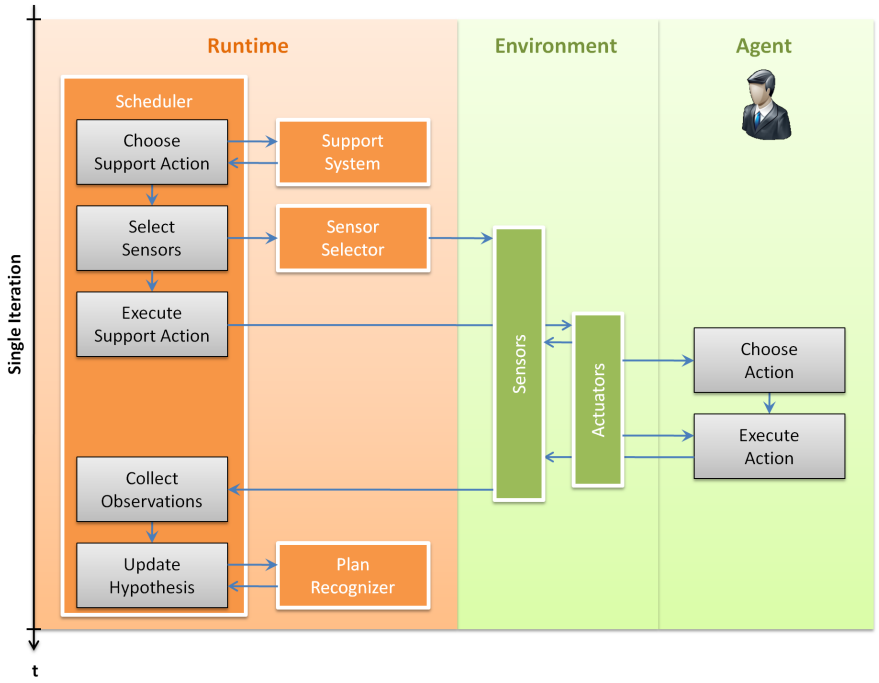


Figure 8.5: Temporal ordering of interactions taking place in RePReTo during one iteration of plan execution/plan recognition.

In the following, it is assumed that all compile time processes have been successfully completed and that the system right now is at the beginning of a new plan recognition iteration n. It is further assumed that the system's current belief about the observed agent's plan is available in form of a plan hypothesis h_n (h_n is called "base hypothesis" in iteration n). In the system's initial iteration, the base hypothesis equals the plan library's initial hypothesis h_0. In all following plan recognition iterations, the base hypothesis equals the updated plan hypothesis from the previous iteration $n-1$ (details on the exact reasoning process are given below).

According to the plan selection and execution model that is presented in section 6.2, one iteration of the agent's plan execution process consists of two steps: The selection of the next action to perform, and the execution of this action. As the system's actions either precede (in the case of providing proactive support) or succeed (in the case of updating the plan

hypothesis) the agent's actions, they have to be performed "between" the agent's iterations (more precisely at the beginning and at the end of an iteration). When determining the concrete order of component invocations, one additionally has to take into account the conditional dependencies that exist between different components with respect to information provision and information consumption. Some components rely on information provided by other components, and the latter ones therefore have to reside further upward in the execution path. These considerations lead to the following sequence of operations, which the system has to perform in this order in every iteration:

1. Choose best support action given current base hypothesis.
2. Activate best subset of sensors given hypothesis and chosen action.
3. Execute chosen support action (if any).
4. Wait for the agent to complete current plan execution step.
5. Collect all resulting sensor observations.
6. Use observations and support action to update current base hypothesis.

The discussion of individual steps starts with step 2: Before proactive support can be actually offered to the observed agent, the system has to "tune" the environment's sensors to optimize the recognition of the expected upcoming (re)action of the agent. As the provision of support might trigger an immediate response of the agent, the system could easily miss possibly important observations if sensor selection has not been finished at the point in time where the support action is executed.

As the expected upcoming action of the agent (and thus the most promising subset of sensors) might depend on the executed support action, this action has to be known to the sensor selector in step 2. For this reason, the system has to decide in advance on the support action that should be executed (step 1). The decision on the best support action only depends on the current base hypothesis, and thus can be optimally made at this early point in time. For the reasons given above, the execution of the chosen action however has to be delayed to the point in time after the selection of sensors finished, and thus is executed in step 3. Recall that the system always has the possibility to choose the special *none* action, which results in no support during the current iteration.

Now, with sensors selected and support (if any) initiated, the system waits for the observed agent to execute its part of the plan execution/plan recognition iteration. The agent's decision on a particular action as well

as the execution of this action thereby might be influenced by the support that is provided. During this phase, active sensors collect observations which might originate from the observed agent's behavior and the execution of the support action. These observations are collected by the scheduler in step 6.

At the end of each iteration, the collected observations are used in step 7 to update the current plan hypothesis and thus to provide the new base hypothesis for the following iteration. Besides the set of collected observations, the plan recognizer also has to take into account the support action that was executed in step 3. The updating of the plan hypothesis ends one iteration of the plan execution/plan recognition process, and the next iteration is started.

8.2.2 Observation Sequence Segmentation Problem

An important question that has not been addressed so far is how the system knows when the observed agent finished his action execution, and thus has to perform the updating of the plan hypothesis with the observations made so far. Unfortunately, there is no simple general answer to this question. If sensor selection is applied, one cannot rely on sensor data to judge whether the agent finished the execution of his action or not. The reason for this is that with the wrong set of sensors "accidentally" selected, one might not be able to collect a sufficient amount of information (or, in the worst case, any information at all). Thus, one needs some other way to decide whether an agent's plan execution iteration has completed or not.

One possible solution to this problem would be to have some external application-dependent component which could provide the plan recognition system with the information that an agent iteration finished (this information will be called *clock signal*, as it allows for the clocking of the plan recognition process). Such an external component could for instance be the global clock in a turn-based system. An example of such a system is a distributed multi-agent system, in which agents communicate in synchronized rounds. Another possibility would be to have some special kind of sensor, which does only allow to observe that some action was executed (the occurrence of a clock tick), but otherwise is incapable of sensing any other information. If excluded from the regular sensor selection, such a sensor could be permanently running and providing the system with clock signals. An example of such a sensor is an acceleration sensor, which can observe that a motion action is executed, but cannot determine the exact action (e.g. its origin and destination).

Another possible solution to the problem of determining the end of the agent-side iteration is to assume a fixed length of each plan execution/plan recognition iteration (clock interval). Whenever this assumption is not justified by the application scenario (which presumably holds for most existing

application scenarios), one has to rewrite the plan library and support model in a way that compensates for this wrong assumption. Practically, this means to allow for actions that span multiple iterations by introducing loops and additional intermediate states in the plan library and support model. Examples of possible patterns to model variable-time-length actions are discussed in section 8.4. One then "simply" uses the greatest common divisor of all possible agent action durations (or some technically reasonable approximation of it) as the assumed clock interval, and rewrites the plan library and support model accordingly.

Both approaches have their specific strengths and weaknesses. While the first approach requires the availability of some external application-dependent component that provides the system with a clock signal and thus is only feasible in specific environments and application scenarios, it does not require rewriting the system's knowledge models. While the second approach is independent of any concrete environment or application scenario, the necessary rewriting of the knowledge models might increase their size and complexity, which in turn might negatively affect the system's compile time and runtime performance.

After all, both presented approaches to deal with variable-length actions are not completely satisfactory under all circumstances. As an investigation of all aspects related to the recognition of variable-time-length actions exceeds the scope of this thesis, this topic is recommended for future research.

8.3 Acquisition of Knowledge Models

A critical aspect of systems which rely on statistical models for reasoning is the availability of correct and sufficient model information. In the case of the proposed RePReTo architecture, this concerns the plan library, sensor model, support model, and cost-reward model. So far it was assumed that these models are given. This section discusses how such knowledge models can be acquired in real-world scenarios.

A naïve approach would be to let a domain expert manually enter the required knowledge into the system. This approach obviously becomes intractable for any non-trivial plan library and application scenario. There are two main reasons for this: Firstly, the required domain knowledge simply might not be available. Even if it is, the sheer amount of states, dependencies, and probabilities to assess might become too large to enter every piece of information into the according models by hand. Thus, other solutions have to be found for the practical acquisition of knowledge models.

8.3.1 Reuse of Design Time Information

If formal methods have been applied in the design and implementation of an environment, then models describing the user's task, the environment's support services, and the existing sensors and actuators in a well-defined way might already be available. Even if in most cases the existing models might not be directly usable by RePReTo (e.g. because their representation is incompatible to the representation that is used in RePReTo, because the existing models are too abstract and not fine-grained enough, or simply incomplete), they might provide a good starting point for assessing the basic structure and dependencies of the required RePReTo knowledge models.

An example from recent research is the GOAL methodology proposed by Stahl in [Sta09]. GOAL is an acronym and stands for "*G*eometry-*O*ntology-*A*ctivity Mode*L*". The GOAL methodology is based on three columns: A detailed geometrical model of the actual environment, an activity model that is based on Activity Theory [Leo78], and an ontology that ties both models together. The methodology allows analyzing typical activities of the user with respect to the surrounding environment with the goal of identifying useful assistance features. Furthermore, the method supports the designer in deciding on the necessary instrumentation of the environment, particularly on which sensors and actuators to use and where to place them. Figure 8.6 shows the central components of the GOAL design method.

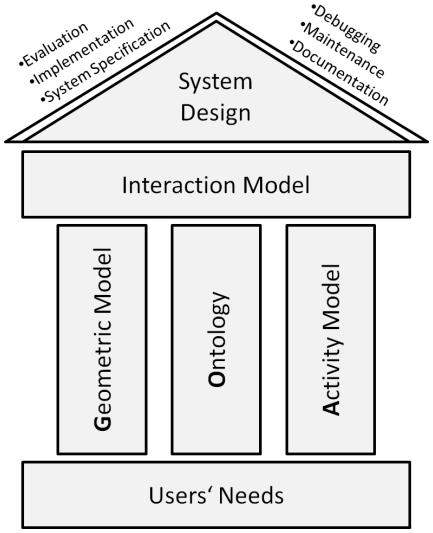


Figure 8.6: GOAL (Geometry-Ontology-Activity Model) Design Method. Source: [Sta09].

Stahl characterizes the role of the models that are defined by his GOAL design method as follows:

> "These models yield the interaction model that describes how the users will be assisted in their activities by the various components of the instrumented environment, with focus on interaction modalities, sensor input, and actuator output."

This sounds very similar to the purpose of the plan library, sensor model, and support model in RePReTo. The rest of this section therefore will have a closer look on how the GOAL models can be used to support the model acquisition process for RePReTo.

GOAL defines the user's behavior in terms of activities. According to the underlying activity theory, activity of any subject is understood as a purposeful interaction of the subject with the surrounding world. Formally, an activity is defined as a triple (*subject, artifact, motive*). Each activity can be hierarchically decomposed into actions and operations. Following this definition, activities are very similar to plans. In fact, every activity can be understood as a plan instance, where the *motive* of an activity equals the plan's goal, and the decomposition of an activity into operations equals the list of actions that realize the associated plan.

Activities are used in GOAL not only to model the behavior of the environment's user(s), but also to represent the system's support services. User and system activities are distinguished by the *subject* component of the according activity triple. A user plan which is supported by one or more system actions then is modeled as an alternating sequence of user and system activities. Figure 8.7 shows an excerpt of a GOAL activity model describing the first steps of an assisted preparation process for bruschetta in an instrumented kitchen environment, which provides assistance through the display of context-dependent textual and spoken preparation instructions. The entities listed in the second to fourth column of the table relate to concepts that are defined in the UbisWorld ontology [Hec05]. Similar descriptions exist for all other (assisted and unassisted) activities that are modeled in GOAL.

As mentioned above, activities are similar to plans. Therefore, the GOAL activity model can be used to derive the basic structure of a RePReTo plan library and support model. This includes possible goals, agent actions, system actions, and the nodes and edges of the plan execution and support graph. One important aspect that seems not to be provided by the GOAL methodology at its current state of development is a statistical model of the user's plan selection and execution process. For RePReTo, such knowledge is important to assess the prior probabilities of plans and actions; both are required to define a valid plan library and support model in RePReTo.

Activity / Action / Operation	Subject	Artifact	HEI	..
Prepare Bruschetta (assisted)				
Breakpoint1	SemanticCookbook			
Gong	SemanticCookbook		Display	
Text and speech	SemanticCookbook		Display	
Highlight items	SemanticCookbook		Display	
Confirmation1	User			
Manual confirmation	User		Control Cube	
Place cutting board, bread knife, and bread on RFID area	User	Bread-knife, Cutting-board, Bread	Antenna1	
Present section 1	SemanticCookbook			
Play video section 1	SemanticCookbook		Display	
Perform section 1	User			
Cut bread in slices	User	Bread-knife, Cutting-board, Bread		
...				
...				

Figure 8.7: Excerpt of a GOAL activity model describing the assisted preparation of bruschetta in an instrumented kitchen. Source: [Sta09].

To circumvent the problem of missing statistical data, one could either extend the GOAL methodology to additionally consider such information in the design process, or one could add the missing information in a secondary step through some process defined outside of the GOAL methodology. Examples for such approaches are the manual addition of the statistical model (with the known drawbacks discussed above), or the automated learning of probabilities through the application of machine learning techniques (the latter approach is discussed in subsection 8.3.2).

The third RePReTo model is the sensor model. As introduced in section 6.3, sensor models describe which observations result from which sensors under which circumstances. In GOAL, sensors are considered to be part of the human environment interface (HEI). Each activity can be associated with one or more HEI components (see fourth column of Figure 8.7). Each element in the HEI field is related to a concept in the UbisWorld ontology, which then might provide a more detailed description of the employed component respectively sensor. One can imagine that one part of this description is a formal model describing the general properties and modes of operation for a specific sensor. Multiple of such models then might be compiled to a global sensor model guided by higher-order information from the GOAL activity model through a process similar to *knowledge-based model construction* [WBG92]. Additional meta information might be provided by GOAL's geometry model, which describes the location of sensors as well as typical places

for the execution of certain activities. Such information can be exploited to refine the sensor model in the case of proximity-based or location-aware sensors. For instance Figure 8.8 shows, that the activity "switch on oven" can only be performed by the user when she is in front of the oven (illustrated by the yellow square in 3D the environment model).

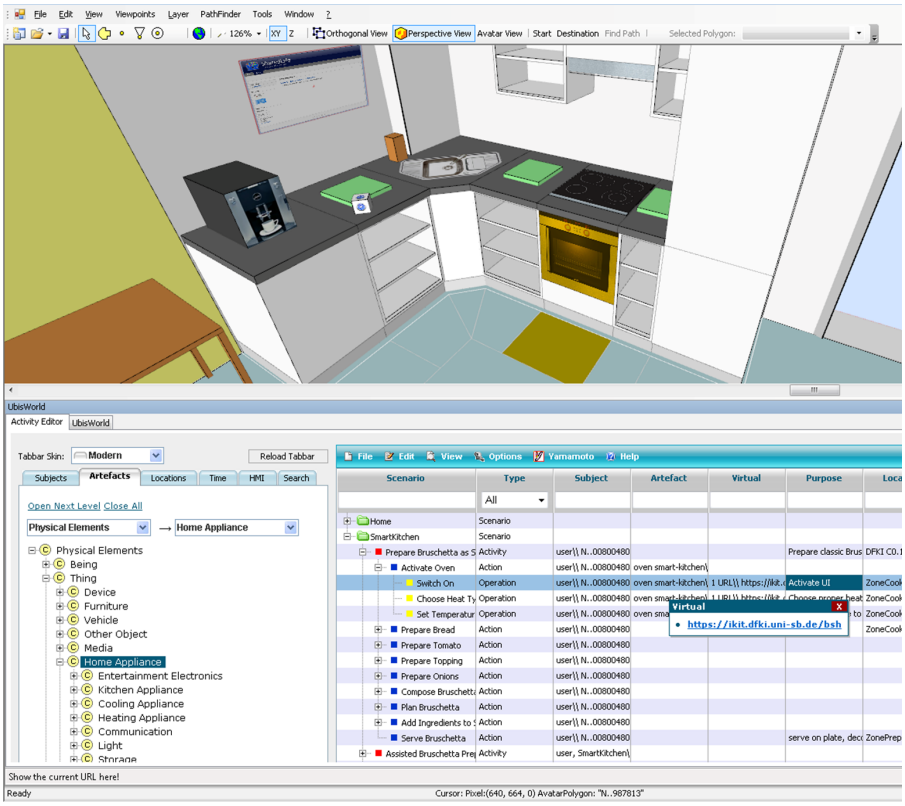


Figure 8.8: Modeling environments, physical sensors, user activity, and system support in the graphical YAMAMOTO editor. Source: [Sta09].

The designer of an instrumented environment is supported in creating the different models that are defined by the GOAL methodology trough a graphical editor (see screenshot in Figure 8.8), which is based on the YAMAMOTO application [SH06]. The lower right side of the screen shows the activity model tabular, which is organized similar to the tabular shown in Figure 8.7. The upper half of the screen displays the geometrical 3D model of the environment, including the assumed positions of artifacts, sensors, and actuators. The lower left of the screen provides the ontology view, which allows browsing the UbisWorld ontology and to insert concepts into the activity model or spatial model through simple drag and drop operations.

8.3.2 Learning of Knowledge Models

If the required knowledge models (plan library, support model, and sensor model) are not available or cannot completely be derived from information collected during the design of an instrumented environment (see previous section), then missing information might be learned from information available from other sources. This section will look into various approaches of utilizing such sources for the acquisition of required knowledge models.

An example of knowledge acquisition from remote sources is the parsing of cooking community websites in order to extract plans (in the form of recipes) that might be executed in a kitchen environment. For this, textual descriptions of recipes have to be translated into a formal representation, e.g. based on an ontological framework. An example of a system with such functionality is the cooking tutor system presented in [MPF+08], which uses a linguistic tool to analyze found recipes and to formulate them using the OntoChef cooking ontology [RBP+06]. Figure 8.9 provides an overview of the main classes which comprise the OntoChef ontology. As one can see from the class diagram, recipes are decomposed into phases, which are then decomposed into tasks, which are finally decomposed into actions.

A collection of recipes now can be understood as a set of plans, where each plan has the goal of preparing a certain recipe. The recipes decompositions into actions equal the associated plan's steps. With this approach, one can derive the basic structure of a plan library (in terms of a plan graph), but it does not provide the required description of the probabilities of choosing particular plans and actions. Later in this chapter it will be discussed how these probabilities can be derived. For now we assume that the a-priori probabilities of all recipes are uniformly distributed.

The previous example described how existing representations of structured processes like recipes can be exploited to assess the general structure of a plan library. This requires that (a) relevant information has been identified and formalized beforehand, (b) the information is available and expressed in some known format, and (c) the plans represented in the information are valid given the potentially new context in which they have to be applied.

With respect to our example of recipe parsing, this means that we can only utilize recipes that (a) are available on the considered website, (b) the parser is able to understand, and (c) are possible to recreate in the target environment (i.e. it is useless to consider baking recipes if there is no oven in our kitchen). As a consequence, the presented approach of exploiting external knowledge sources is not feasible to recognize plans and variants in completely unknown domains, or to adapt plans with unmet preconditions to new contexts, regardless of the source of information.

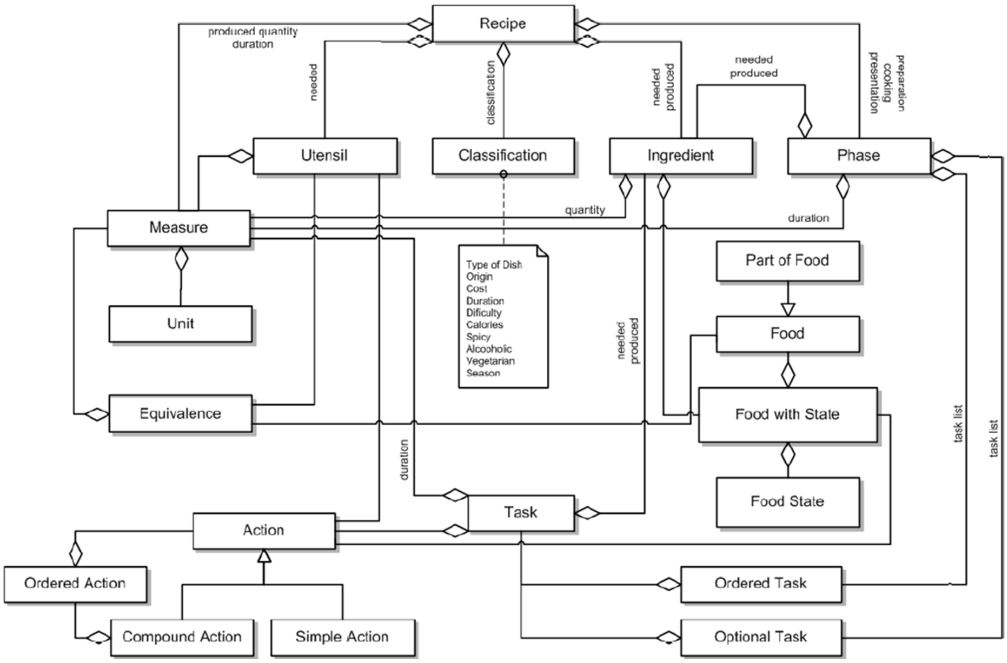


Figure 8.9: Main classes of the OntoChef ontology to represent cooking plans. Source: [RBP+06].

One possibility to learn about unknown plans and plan variants in general domains is to observe and record how users interact with an environment and the applications installed therein. With a sufficient amount of sampled information available, one then might be able to apply machine learning techniques in order to automatically derive matching plan libraries and support models. The rest of this section presents system's which allow to record the user's actions in an environment, and concludes with the description of existing approaches to learn plan libraries from observation data.

One of the early systems which tried to observe an agent's actions in physical environments is SPECTER [SBK05a, KHW06, BKSB08, KJSB08], which aims to automatically create an episodic digital memory (the so called "personal journal") of everyday activities for its user. The purpose of the personal journal is to augment the user's biological memory through data captured in an intelligent environment. Different abstraction steps are applied to infer higher-order information from observed lower-order sensor data. The resulting memory contains interactions between the user and the environment, interactions between the user and support applications in the environment, and even interactions between the user and SPECTER itself.

The resulting episodic memory can then be exploited by the user to review past activities in a process called introspection or to receive decision support that is based on the user's previous experiences and is provided in the form of so called *recomindations* (sic!) [PBK+06]. The user's digital memory might furthermore be accessed by adaptive systems to learn about the user's habits and preferences, or to construct a general user model over time [KHS06]. Figure 8.10 presents the basic functional blocks of the SPECTER system and their fundamental interrelations. One particular interesting aspect of SPECTER with respect to the learning of plan libraries, support models, and sensor models, is its ability to capture the user's context while observing her behavior. This additional information allows for the creation of more fine-grained models than for instance in the case of recipe parsing from a website, where virtually no context information is provided.

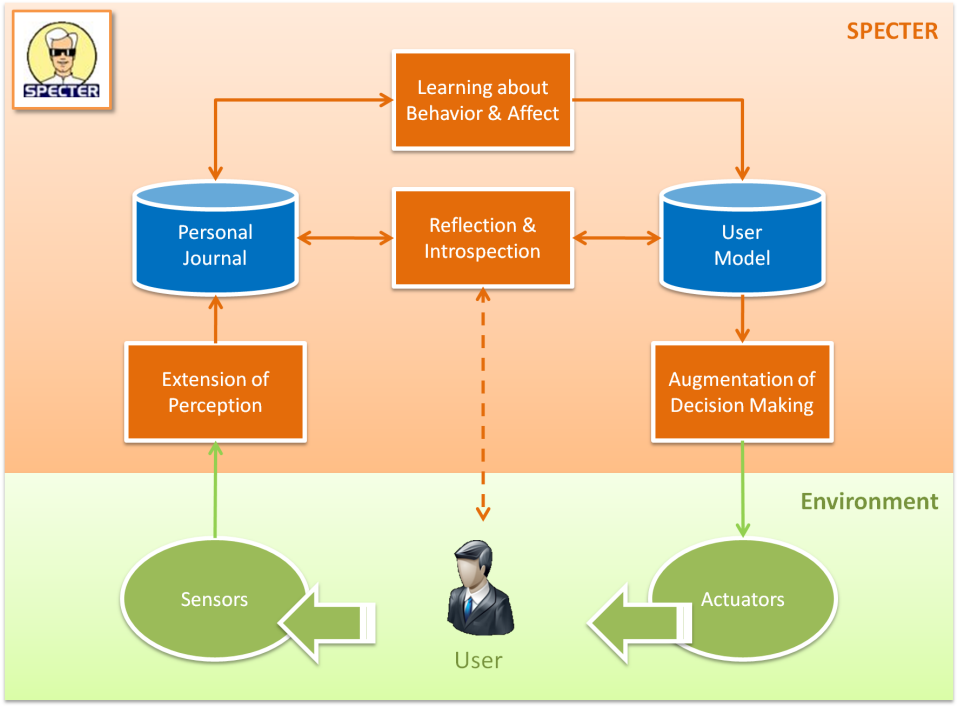


Figure 8.10: Conceptual architecture of the SPECTER system.

A generalized variant of SPECTER-like interaction logging systems are so-called *Open Personal Memories* [SKW06]. An open personal memory is a collection of past events that is "open" in the sense that arbitrary sources can contribute to the collection and "personal" in the sense that the memory is owned by a single user who has full control over the use of stored

information. Such a memory could be implemented on a mobile device that is under the user's direct control like a Smartphone, or – if network access is available in the environment – could be realized as a web service hosted on a trusted server. Every time a user interacts with the environment, the environment or contained applications can contribute high level descriptions of the interactions to the user's open personal memory.

The single-user scenarios of SPECTER and open personal memories are extended to a multi-user scenario in SharedLife [JKBS07, MBS+07, KJSB08, WKH06]. In addition to information from the user's own personal digital memory, SharedLife allows to locate and access relevant information that is stored in other user's digital memories [KBSM07, MBKJ08]. An important focus of SharedLife lies on privacy-aware sharing of digital memories, with the ultimate goal of supporting communication between individuals and learning from the experiences of others in a way that does not conflict with individual users' privacy constraints.

One of SharedLife's demonstration and evaluation scenarios is the smart kitchen environment, which was developed in the context of this thesis and is presented in chapter 9. It is a good example of a system that supports the learning of new plans. As we describe in section 9.2 the smart kitchen's central application – the *Semantic Cookbook* – has a special recording mode, in which the user can record a recipe that was previously unknown to the system. When in recording mode, the system captures all (unstructured) sensor events that occur and associates them with the newly recorded recipe. This recipe can then be shared with other users via the Internet. The specific strength of SharedLife with respect to the learning of plan libraries lies in this ability to share possible training data between multiple users, and thus to broaden the set of training instances that are available to an eventually applied machine learning approach. This allows the system to consider a broader range of plan variants in the learning process, which without such sharing approach might not have been available in the recordings of a single user.

Although the presented systems like SPECTER, open personal memories, and SharedLife are able to create an extensive log of an agent's actions (depending on the concrete instrumentation of the environment with sensors), their data can often not be directly used to derive the knowledge models which are required for plan recognition. The main reason for this is the missing higher-order structure that plans impose on otherwise unrelated sequences of actions. This structure defines the nature of a plan and distinguishes goal-directed from undirected behavior and allows to map observations to plan hypotheses. For unknown plans this higher-order structure in the first place only exists as a mental model that the executing agent has in

mind, but can generally not be directly observed through any kind of sensing technology that exists today.

In order to discover the higher-order structure of unknown plans in sequences of collected observations, one has to search for repeating patterns within these observations. The rationale behind this is, that random behavior is unlike to produce the same (or a sufficiently similar) sequence of observations several times. Actions guided by some underlying plan are executed more or less in the same way and sequence every time the associated goal is pursued. This fact is exploited by Bauer in [Bau99], where he proposes an approach to the automatic acquisition of hierarchical plan libraries from sample action sequences. In particular, he introduces a *clustering algorithm* that allows groups of "similar" action sequences to be discovered and used for the generation of plan libraries. Bauer's approach does not rely on the existence of labeled actions or formalized domain knowledge, although it can utilize such information to improve its results.

After the higher-order structure of the plan library has been learned from the sample data, the associated statistical model can be acquired by simply "counting" the relative occurrence of particular plans, actions, and resulting observations. If the set of sample observations which is used for learning is very small, the resulting statistical model might significantly deviate from the true probabilities of plans, actions, and observations. However, this is most probably also the case if the according models have been carefully handcrafted by some domain expert. The main reason for this is, that each agent has its particular habits and preferences, which are impossible to capture in a general model. Thus, the probabilities of every model (regardless if learned or otherwise given) should be iteratively refined over time as more (individual) sample data becomes available. In [Bau99] Bauer even argues that learned plan libraries can be expected to support the plan recognition process particular well as they contain abstractions of actual agent's behavior rather than idealized plans designed by a knowledge engineer.

As mentioned earlier the application of machine learning techniques for the acquisition of plan recognition knowledge models requires the availability of sufficient amounts of training data. Today, the lack of observation data is one of the main reason that hinders the application of machine learning techniques as described above. Hence, the acquisition of extensive activity copora is an important topic for future research (see section 11.3).

8.4 Design Patterns for Plan Libraries

The theoretical plan selection and execution model that we presented in section 6.2 and used in the previous chapters as foundation for our state-aware plan recognition approach and decision-theoretic utility model provides an abstract framework for the description of agent plans.

This section gives examples how some typical situations that arise in real-world plan libraries can be represented in the presented state-based plan library. Due to the general nature of the described approaches, they can be considered as reusable design patterns, which can be applied to ease the manual development or automated composition of plan libraries.

8.4.1 Delayed Execution

The state-based plan selection and execution model that we presented in section 6.2 assumes that in each time step the agent selects and executes exactly one action from its action repertoire. This implies that each action immediately follows its predecessor action and that there exist no delays or pauses in the plan execution process.

In practical applications the execution of plans without delays in general is not a reasonable assumption. For instance an agent might not right away start with the execution of a plan, but might spent some time in the *idle* state beforehand. Reasons for delays during the execution of plans include the agent's waiting for some effects of its actions to occur, the agent's being distracted from plan execution, time required for (re)planning further steps, or simply the agent's laziness. The duration of such delays might vary, and often we do not know the exact duration in advance, but can only give a probability distribution over expected durations of delay.

The *delayed execution pattern* that we present in the following can be applied, if the length of a delay is distributed according to a *geometric distribution*. Geometric probability distributions (and exponential distributions as their continuous pendants) are often used to describe the length of time between the occurrences of two events. In our case, these events are the execution of actions from the agent's action repertoire. In the following, we assume that in each time step a delay occurs with probability p, and that plan execution continues with probability $(1-p)$. The cumulative probability that a delay of exactly n time steps occurs then is

$$p^n(1-p)$$

A geometric distribution is *memoryless*, which means that the probability that plan execution continues in the next step is always $(1-p)$, indepen-

dent of the length of the delay so far. This property allows for the efficient and straight-forward representation of geometrically distributed delays in our plan selection and execution model: The delayed execution pattern models delays in the plan library by introducing a special-purpose *none* action to the agent's action repertoire. This action represents the case where the agent performs no regular action and if "executed" results in no state transition and no sensor observations.

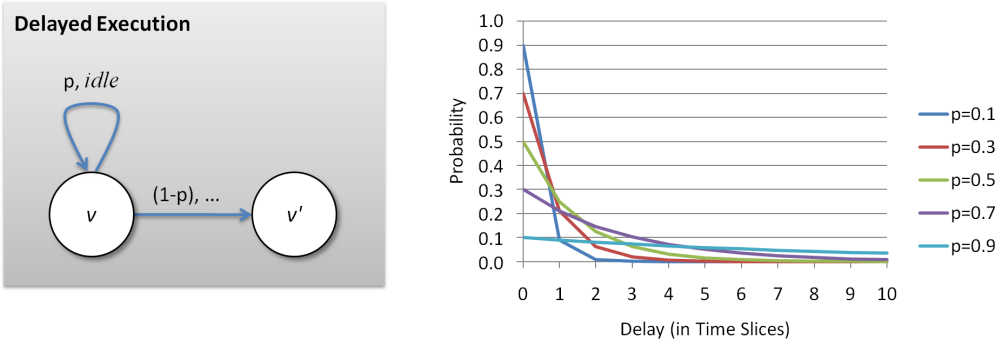


Figure 8.11: Delayed execution of actions modeled through loops in the plan graph (left), and the resulting geometric distribution of delay times (right).

In the plan graph this behavior is represented by the introduction of additional edges which are labeled with action *idle*, occur with probability p, and loop back to the originating source state. The left side of Figure 8.11 shows an example of such an edge for an exemplary node v. Probability p denotes the likelihood, that a delay occurs in the next time step[2]. Note that each node in the plan graph can be assigned an individual probability of delay. The probabilities over the total length of the delay that result for different exemplary values of p are shown on the right side of Figure 8.11.

8.4.2 Unpurposeful Actions

In section 3.6 we mentioned *purposeful acting* as one of the standard assumptions of plan recognition. This assumption states that all actions which the agent executes serve the purpose of the currently followed plan.

Similar to the assumption of uninterrupted plan execution that motivated the delayed execution pattern presented in subsection 8.4.1, the assumption of purposeful acting is not always reasonable in real-world applications. Plan

[2]Accordingly, the designer of the plan library has to make sure that the probabilities of outgoing "regular" edges are chosen such that they sum up to $(1 - p)$.

execution might be interrupted by actions which do not belong to the currently executed plan for multiple reasons. Such reasons include fill-actions which are executed to bridge a delay during the execution of a plan, or intermediate short-term tasks that interleave the execution of a plan, like answering a phone call during cooking.

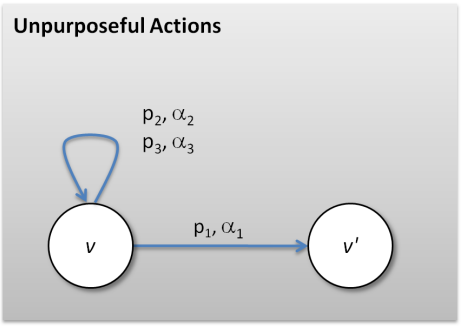


Figure 8.12: Actions which neither support nor hinder the execution of a plan can be considered as a special case of execution delay.

The unpurposeful action pattern is similar to the delayed execution pattern, which also becomes evident by the similar appearance of the library extracts shown in Figures 8.11 and 8.12. In fact, the execution of an unpurposeful action for the plan recognition system simply is a delay in the execution of the current plan. As the action serves no purpose with respect to the currently executed (and thus recognized plan) the system only has to distinguish between the "execution" of an *idle* action some particular unpurposeful action with respect to the associated probabilities.

Figure 8.12 shows an exemplary excerpt of a plan library which accounts for the execution of two unpurposeful actions α_1 and α_2 with probabilities p_1 respectively p_2 in state v. α_1 in this example is a purposeful action which continues the execution of the plan. Probabilities for all actions possible in state v have to be chosen such that they sum up to 1 (in the example this requires that $p_1 = 1 - p_2 - p_3$).

8.4.3 Long-Duration Actions

The plan selection and execution model presented earlier assumes that exactly one agent action is executed in each time step. Depending on how a time step is defined (see discussion in section 8.2) it might be required to consider actions whose execution spans more than one time step. Such actions are called *long-duration actions* in the following.

If the execution duration of an action follows a geometric distribution (which means that a long-duration action terminates in the next time step with a probability of $(1 - p)$ which is independent of the action duration so far) then a variant of the delayed execution pattern that we described above can be used. Often the execution duration of actions is not distributed following a geometric distribution but some other distribution like the normal distribution. The *long-duration action pattern* that we present in the following can be used to model actions with a length that is distributed following arbitrary probability distributions which have a finite and known maximum length.

As the geometric distribution is the only memoryless probability distribution that exists, it is the only distribution which can be represented through a simple loop in the plan library. To represent any other distribution we have to introduce additional intermediate states, which are used as *memory states* that implicitly "count" the number of time slices that passed so far [3]. The general idea is to represent a transition $v \xrightarrow{\alpha} v'$ that spans n time slices by the longer transition sequence $v \xrightarrow{\alpha} v'_{n-1} \xrightarrow{\alpha} \ldots \xrightarrow{\alpha} v'_1 \xrightarrow{\alpha} v'_0$, where v'_i is a virtual intermediate memory state which represents the case that action execution continues for i more time slices (in order to simplify the notation we rename the destination state from v' to v'_0).

The number of required virtual intermediate states is one less than the maximum duration. Transitions then occur as follows: From each intermediate state v'_i (with $0 < i < n$) a deterministic transition $v'_i \xrightarrow{\alpha} v'_{i-1}$ occurs with probability 1. Thus, intermediate states simply "count down" the remaining time steps. The total length is determined by the initial transition from state v to one of the states v'_i: If p_i is the probability that the action execution takes exactly i time steps, then transition $v \xrightarrow{\alpha} v'_{i-1}$ occurs with probability p_i. No other transitions occur for action α.

The long-duration action pattern is illustrated by Figure 8.13. On the left side the known case of a short-duration action α which spans exactly one time slice is shown, while on the right side of Figure 8.13 an example of the same transition with a long-duration action α of maximum length 4 is shown. In this example, α spans one time slice with probability p_1, two time slices with probability p_2, and so on.

The long-duration action pattern models actions that have a non-geometric distribution of duration at the cost of the introduction of additional states and transitions in the plan library. On the downside, this can lead to a

[3] As these intermediate states are not part of the original plan execution state space, but are only introduced to help us modeling the execution of long-duration actions, they are called *virtual intermediate states*.

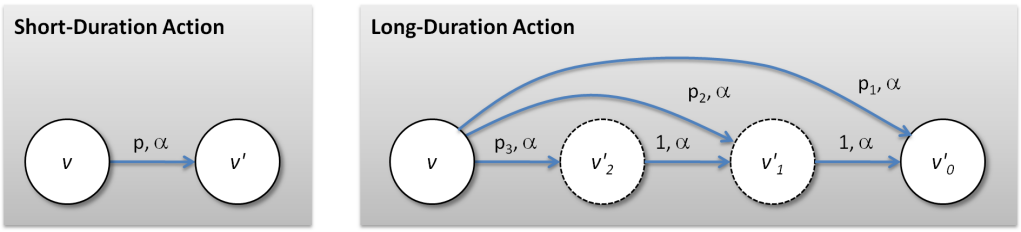


Figure 8.13: While *short-duration actions* always span exactly one time slice (left), *long-duration actions* can span multiple time slices through the introduction of virtual intermediate states (right).

blow-up of plan library size, especially if multiple long-duration actions with a possibly large maximum duration have to be represented in the library. On the upside, the negative impact is alleviated by the fact that the resulting additional node and transition structures are extremely sparse, which only leads to a moderate increase in overall model complexity. Implementations can exploit this fact by applying optimization techniques for sparse structures that limit the negative impact on memory consumption and evaluation time. An example is the use of adjacency lists for the representation of the plan graph in our REPRETO system.

8.4.4 Common Prefixes and Suffixes

A straight-forward way of representing a set of n plans with lengths $l_1, \ldots, l_n$ is through a plan library with $\sum_n^{i=1} l_i$ states (plus the obligatory *idle* state), where $v_{i,j}$ denotes the state which represents the successful completion of the step j in plan i, and where states v_{i,l_i} are terminal states for $1 \leq i \leq n$. The top half of Figure 8.14 shows the resulting plan graph for an example library that represents three plans of length four.

The *common prefixes and suffixes* pattern under certain circumstances allows for a more compact representation of plan libraries if groups of plans in the library share (partial) common prefixes or suffixes. Plans have a common prefix if they start with the same sequence of actions. Plans have a common suffix if they realize the same goal and end with the same sequence of actions. In the example from Figure 8.14 all plans share the common prefix $[\alpha_1]$, and the first and second plan share the common prefix $[\alpha_1, \alpha_2]$. Furthermore the second and third plan share the common suffix $[\alpha_3]$ if we assume that goals g_2 and g_3 are equivalent. Under certain conditions we can merge intermediate plan execution states which belong to the same common prefix/suffix, which reduces the number of states and transitions in the resulting plan library.

Recall from section 6.1, that intermediate states represent "configurations" of the environment and the agent which result from the execution of actions. If the same sequence of actions leads to the same sequence of "configurations" (which also implies that the same sensor observations occur), as well as to the same costs and rewards, then these matching states can be represented by a shared prototypic set of plan library states. For instance this parallelism exists whenever the effects of actions solely depend on the physical state in which the actions are executed, but not on the pursued goal.

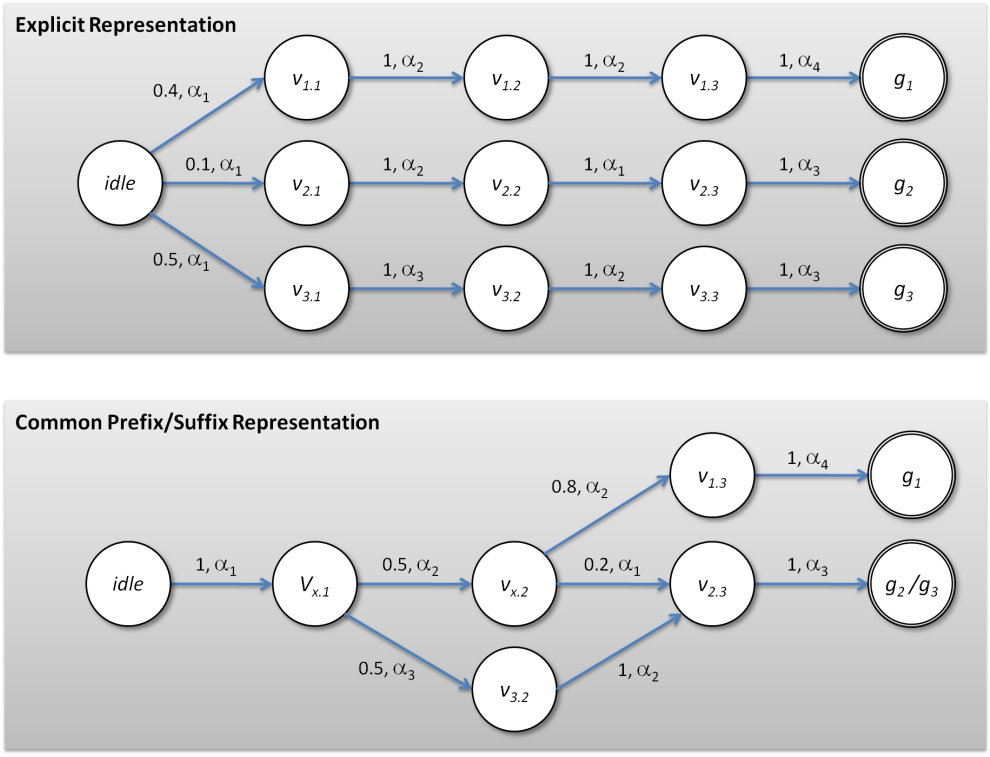


Figure 8.14: Multiple plans with a partial common prefix and/or suffix (top) can be combined to a merged prefix/suffix state sequence (bottom).

Merged prefix state sequences also allow for the representation of parts of plans which only match (continuous) subparts of a common prefix/suffix. The bottom part of Figure 8.14 illustrates how this pattern is applied. In the example, the merged state $v_{x.1}$ replaces the old states $v_{1.1}, v_{1.2}$, and $v_{1.3}$ for the (partial) common prefix $[\alpha_1]$. Merged state $v_{x.2}$ replaces the old states $v_{2.1}$ and $v_{2.2}$, and together with $v_{x.1}$ represents the common prefix $[\alpha_1, \alpha_2]$. Merged state $v_{x.3}$ replaces the states $v_{2.3}$ and $v_{3.3}$, and represents the shared suffix $[\alpha_3]$ (recall that we assumed that g_2 and g_3 are equivalent).

Whenever the structure of a plan library is modified, we have to make sure that we adjust the probabilities associated to individual edges in way that preserves the original probabilities of the contained plans, plan variants, and goals. In the example library shown in Figure 8.14 this is realized by representing the probabilities associated with the edges that leave the *idle* state in the original library (top) through probabilities assigned to the edges leaving the merged nodes $v_{x.1}$ and $v_{x.2}$ in the compacted version of the library (bottom). For common suffices we do not have to adjust the given probabilities, as plan execution "converges" on suffixes, which happens independently of the transition probabilities which rather describe "outgoing" edges.

One could be temped to also represent common infixes of plans by shared state sequences, but this is not possible due to the representation as a Markov process and the resulting requirement of memorylessness. Imagine two plans $[\alpha_1, \alpha_2, \alpha_3]$ and $[\alpha_3, \alpha_2, \alpha_1]$. If both would use a shared infix $[\alpha_2]$ it would be impossible after the execution of α_2 to decide with which plan to continue, as there is no way of remembering whether execution started with the first or the second plan. Hence, common infix plans can only appear as parts of common prefixes or common suffixes, but never "standalone".

8.5 Summary

In this chapter we discussed topics related to the practical application of the proposed models and methods. We started with a presentation of the RePReTo architecture, which denotes a set of knowledge models and software components which together provide plan recognition, resource-aware sensor selection, and decision support capabilities to applications which want to realize proactive support services in instrumented environments. We described the components which are required to realize the proposed functionality, their relation to the theoretical models and methods that we defined in previous chapters, and the interaction of these components with respect to the flow of information and existing temporal dependencies.

The resulting RePReTo tool set is provided as a Java library for the integration into existing and future software systems. It is applied in our *Smart Kitchen* environment (see chapter 9) and was used to perform the empirical evaluation study that we present in chapter 10.

9 Application Scenario Assisted Cooking

This section describes the implementation of an *instrumented kitchen environment*, which was realized in the context of this thesis as a *testbed and demonstration scenario* for the application of plan recognition techniques in instrumented environments. The presented environment makes use of our RePReTo system (see chapter 8) to realize plan recognition, sensor selection, and proactive support services.

The chapter comprises two sections. Section 9.1 introduces the *Smart Kitchen* environment and gives an overview about applied sensors, actuators, and other general technical infrastructure. Section 9.2 introduces the *Semantic Cookbook* application, which uses the *Smart Kitchen* environment to support the user in the preparation of food. We present the *Semantic Cookbook*'s functions and explain how plan recognition is applied by the application.

9.1 Smart Kitchen Environment

In this section we describe the technical setup of the *Smart Kitchen* environment [Sch07a] (see Figure 9.1). The *Smart Kitchen* is located at the German Research Center for Artificial Intelligence (DFKI) in Saarbrücken and has the purpose of supporting its user in the preparation of meals.

The kitchen is equipped with a multitude of sensors to observe the user and her behavior. Several actuators allow for the provision of visual or audible assistance and the control of kitchen appliances. Sensors and actuators are controlled by the *Semantic Cookbook* application, which implements the kitchen's user support functionality and is introduced in section 9.2.

Figure 9.1: The *Smart Kitchen* environment is used as a testbed for plan recognition in instrumented environments.

9.1.1 Sensors

In order to observe the user's actions, the *Smart Kitchen* environment is instrumented with several different kinds of sensors (see Figure 9.2):

- **Wide-Angle Video Cameras:** Two cameras record an audio-video stream of the cooking process. The video is used to record spoken instructions, as well as preparation actions that cannot be observed by other sensors. The cameras are placed above key locations like the stove and countertop (see Figure 9.2, top left). Camera data is processed by a differential picture analysis computer vision algorithm to detect the degree of activity that is captured by each camera. The system then automatically switches to the camera which shows the most amount of motion and hence is assumed to record the current main activity. Video information from all cameras is kept, such that at a later point in time other perspectives might be chosen by the user or the system.

- **Radio Frequency Identification (RFID):** RFID is used to unobtrusively identify and locate ingredients and tools in the *Smart Kitchen* en-

vironment (a detailed description of RFID is given in subsection 2.3.2). For this purpose, all movable objects are equipped with passive RFID transponders that hold a unique identification number (see Figure 9.2, top middle). Five RFID antennas are mounted under the countertop and allow to detect if an ingredient or tool is placed on or removed from the surface.

- **Digital Scale:** A digital scale is used to measure the amount of ingredients (see Figure 9.2, top right). Software applications can access the scale's measurements via a serial interface. The scale just records the weight of an object, while the identity of the object is determined via RFID. This information then is fused with the weight measurements by the environment's software infrastructure.

- **Networked Kitchen Appliances:** Several appliances in the kitchen are equipped with powerline interfaces, which allow to remotely query the appliances' current state (see Figure 9.2, bottom left). Accessible information is: Power level and presence of cookware on each inductive hot plate (stove), mode of operation, air circulation, timer settings, and temperature level (oven), selected program and timer information (dishwasher), power level and light status (hood), and temperature setting and door-closed status (fridge and freezer).

- **Wireless Sensor Network Nodes:** The use of manual kitchen tools can be observed by attaching wireless sensor network nodes to them (a detailed description of sensor networks is given in subsection 2.3.1). An example of such a tool is the instrumented whisk (see Figure 9.2, bottom middle), which has attached a *MICAz* [Cro09b] node that is equipped with a 3-axis accelerometer. The accelerometer's data is used to detect if the whisk is used, for how long the whisk is used, and the intensity of use (in terms of applied energy). The raw sensor data is locally evaluated on the node, and the resulting abstracted information then is forwarded via a gateway to the kitchen's software infrastructure.

- **Tangible Control Cube User Interface:** The control cube device (see Figure 9.2, bottom right) provides a tangible user interface for the control of the kitchen's main functions and is made of dirt-resistant plastic. With a side length of 9 cm it is big enough to be used with sticky or greasy hands. Each of the six sides of the cube is assigned a distinct function that is represented by an icon printed on the according side. A function is invoked by turning the according side of the cube

upward. The cube's orientation is determined by an accelerometer, which is attached to a MICAz mote that is embedded into the cube.

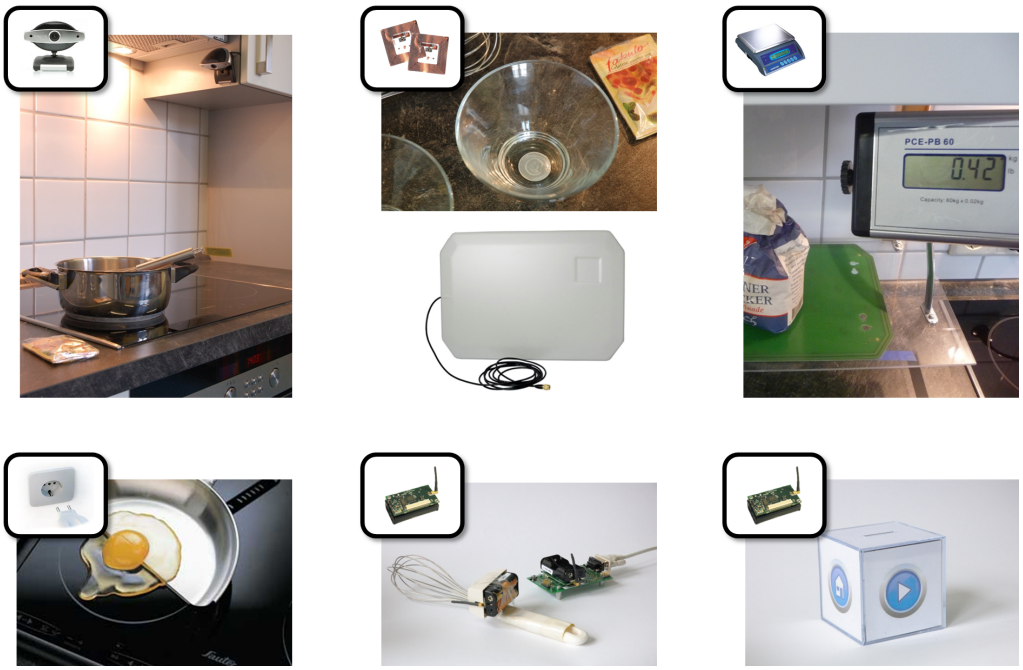


Figure 9.2: Sensors of the *Smart Kitchen*: Cameras (top left), RFID reader (top center), digital scale (top right), networked appliances (bottom left), instrumented tools (bottom center), and control cube interface (bottom right).

9.1.2 Actuators

In order to realize support services, applications must be able to influence either the behavior of the user (e.g. through the provision of helpful information), or the state of the environment (e.g. through the operation of appliances on behalf of the user). For this purpose, the *Smart Kitchen* environment is equipped with several actuators:

- **Large-Screen Display:** A 30" flat screen display is mounted above the kitchen countertop (see Figure 9.3, top left) and is used to provide visual information to the user, e.g. by displaying information on the environment's current state, instructions on upcoming actions, or context-dependent information on the objects that are used. The display for instance shows the graphical user interface of the *Semantic Cookbook* application, which we present in the following section.

- **Loudspeakers:** Loudspeakers (see Figure 9.3, bottom left) are used by applications to provide acoustic information and assistance to the user. Acoustic information might be used for a variety of reasons: On the one hand, it allows for the ambient and thus unobtrusive notification of users through so-called auditory icons [Jun08, Jun09]. On the other hand, explicit acoustic commands or alerts might support the graphical presentation, especially in situations where the user is unable to constantly follow the information presented on the screen, e.g. because she has to focus on performing a complex preparation step.

- **Networked Kitchen Appliances:** The powerline interface of networked kitchen appliances can not only be used to query the current state and settings from each device (see previous subsection on sensors), but also to remotely adjust device settings and to trigger special functions (see Figure 9.3, right): The hood's fan level and lights can be adjusted, the temperature of the fridge and freezer can be changed, and finally the oven's temperature setting can be altered remotely.

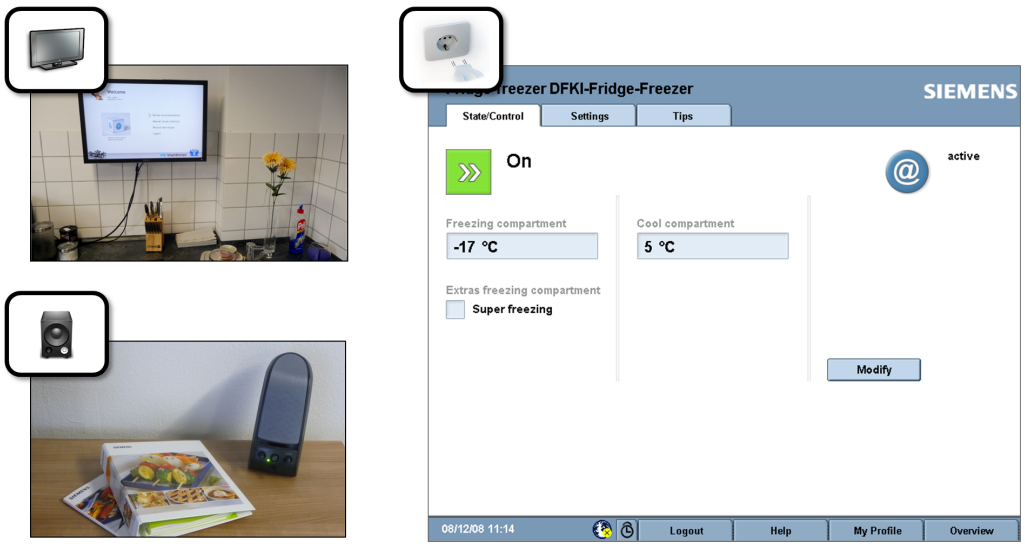


Figure 9.3: Actuators of the *Smart Kitchen*: Large-screen TFT display (top left), loudspeaker (bottom left), and remotely controlled networked kitchen appliances (right, screenshot of control GUI).

9.1.3 Technical Infrastructure

Sensors and actuators are accessed, controlled, and orchestrated by a complex hardware and software infrastructure. This infrastructure has the purpose of proving applications that are deployed in the environment with the necessary interfaces to access the available sensors and to utilize the available actuators. Figure 9.4 gives an overview of the components which comprise the technical *Smart Kitchen* infrastructure.

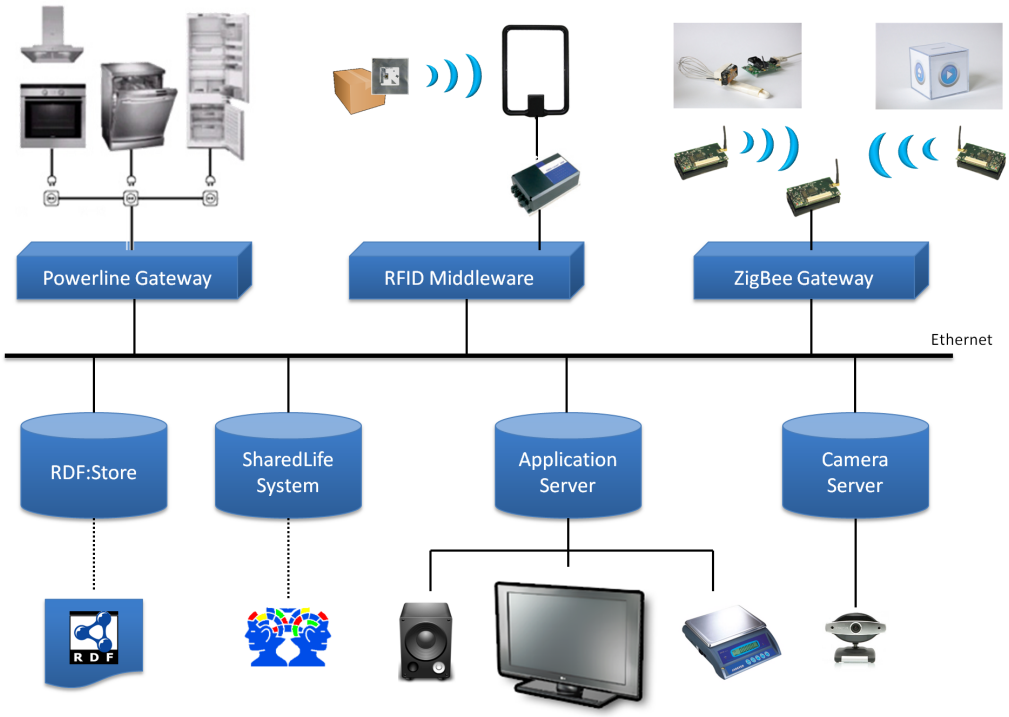


Figure 9.4: Technical infrastructure of the *Smart Kitchen* environment.

- **Powerline Gateway:** The powerline gateway connects devices with a *powerline network* interface to the *Smart Kitchen* infrastructure. This interface allows to query and control devices via the regular electricity network. In the kitchen, several powerline-enabled appliances are installed: Oven, stove, hood, fridge, freezer, and dishwasher.
- **RFID Middleware:** The RFID middleware centrally coordinates the activity of RFID readers in the *Smart Kitchen* environment. It has three main purposes: It synchronizes the reading intervals of readers with overlapping antenna fields to prevent recognition errors caused by

interference (see subsection 2.3.2 for a discussion of the problem of interfering RFID fields), it filters the RFID event stream for sporadic false negatives, and it provides an abstraction from single RFID transponders to whole objects (a single object might be tagged with multiple transponders). The middleware is based on the RFID middleware that was developed by the author of this thesis for the *VirtualConstructor* [NGK+05] exhibit and the IRL environment (see subsection 5.1.3).

- **ZigBee Gateway:** The ZigBee gateway connects nodes of the wireless MICAz sensor network (see subsection 2.3.1) to the rest of the *Smart Kitchen* infrastructure. ZigBee is a communication protocol for small, low-power digital radios. ZigBee is targeted at radio-frequency applications that require low data rate, long battery life, and secure networking. In the *Smart Kitchen* environment, ZigBee is used to communicate with tools like the instrumented whisk or the control cube interface.

- **RDF:Store:** An RDF:Store [Sch06] serves as the environment's central exchange hub for knowledge models. Sensors like the RFID antennas, the kitchen appliances, the digital scale, or the wireless sensor network nodes represent their observations as RDF [MM04] models and advertise them to the RDF:Store. Other components of the environment or applications hosted in the environment can query the RDF:Store for relevant models via a pull mechanism, or can use a push mechanism to be notified whenever new information is available. SPARQL [PAG06] queries can be used as semantic filters whenever an information consumer is only interested in partial data.

- **SharedLife System:** SharedLife is a system that allows for the capturing, introspection, sharing, and exploitation of digital collections of personal user experiences, opinions, and preferences [MBS+07, KJSB08]. In the *Smart Kitchen* environment, sensor information describing the user's actions is forwarded to the SharedLife system and stored in the user's *digital memory*. These experiences then might be shared with other users [KBSM07, JKBS07] (see subsection 2.3.4). Vice versa, the user itself can access other user's memories, e.g. in order to learn about food preferences or allergies of invited guests (see "automated triggering of sharing requests" in subsection 9.2.3).

- **Application Server:** The application server has two purposes: On the one hand, it hosts the infrastructure's background services, which coordinate the interplay of different infrastructure components and query passive sensors like the electronic scale. On the other hand, the server

hosts the applications which use the environment's infrastructure to provide assistance to the user. An example of such an application is the *Semantic Cookbook*, which is presented in the following section.

- **Video Server:** The video server is responsible for the synchronous capturing and replay of the multi-angle video stream that is recorded by the cameras in the *Smart Kitchen* environment. Because this involves the processing and storage of huge amounts of data, the video server is implemented as a dedicated service. The video server's operation can be remotely controlled by applications via a network interface, which also is used to access the image data of recorded/replayed video streams.

9.2 Semantic Cookbook Application

The *Semantic Cookbook* application [Sch07a] is hosted by the *Smart Kitchen* environment that was presented in the previous section. The application supports its user in preparing meals through the automated creation, sharing, and exploitation of semantically annotated cooking videos.

The *Semantic Cookbook* application supports two basic modes of operation, which utilize the environment's infrastructure in different ways: The so-called *observation mode* uses the environment's sensors to passively follow the user's cooking process and to create digital records of the user's personal recipes (see subsection 9.2.1). The so-called *instruction mode* exploits existing records in order to assist the user in recreating own or other users' recipes. Here, the environment's sensors are used to monitor the user's progress, to notify her on deviations from the recorded recipe, and to assess the user's current context in order to provide situation-aware support, e.g. on the usage of kitchen tools (see subsection 9.2.2). The *Semantic Cookbook* applies plan recognition in the instruction mode to assist the user in the selection of recipes and to automatically gather guests' food preferences and allergy information by issuing according SharedLife requests (see subsection 9.2.3).

9.2.1 Observation Mode

In *observation mode*, the *Semantic Cookbook* application passively follows the user's regular cooking process through the environment's sensors. Of special interest are state changes of kitchen appliances and the presence and absence of food items and kitchen utensils at key locations like the countertop. The user might additionally mark important key points during the preparation by setting so-called "breakpoints". At such breakpoints, a complete snapshot

of the environment's current state is taken. This information can be used in the instruction mode which is described below to synchronize recorded and actual progress. After filtering and further processing, the resulting events are aligned with the recorded video streams via their timestamps. Both, video and event streams together form a digital recording of the prepared recipe. This way, the *Semantic Cookbook* application over time creates a digital collection of the user's personal recipes. This collection – in parts or as a whole – might then be shared with other users over the Internet.

Figure 9.5 shows the main screen of the cookbook application as it is presented to the user on the kitchen's display during observation mode. The screen is divided into three areas with distinct functionality:

Figure 9.5: Main screen of the *Semantic Cookbook* application showing the progress during the preparation of Red Wine Mousse in *observation mode*.

- **Video Panel:** The left third of the screen is occupied by the video panel. The panel displays images of the live video streams that are recorded by the cameras in the *Smart Kitchen* environment. For each camera there exists a small preview window in the bottom row. The big video window repeats the image of the selected "main" camera. The main camera is automatically and continuously chosen from the set of available cameras through a motion detection algorithm, which identifies the camera image that currently shows the main activity.

- **List of Ingredients and Tools:** The center of the screen shows a list of ingredients and tools that have been used so far. Whenever an object appears for the first time, it is added to the list and a matching semantic annotation (object's class, amount, and time of use) is added to the video. Kind and number of used ingredients and utensils are observed by the RFID antennas under the kitchen's countertop. The list is ordered according to the time of appearance of the according objects. Ingredients that are left over at the end of the cooking process are ex post removed from the list and semantic annotations.

- **Context-Dependent Assistance:** The right side of the screen is occupied by the information panel. Besides the name of the prepared dish, additional information about the currently used object is displayed. This for instance includes directions on how to use certain kitchen tools, product information like nutrition facts, or concrete warnings in the case of expired or damaged ingredients. Concrete examples of context-dependent assistance are given in the following description of the *Semantic Cookbook*'s instruction mode.

During the actual cooking process the system requires no direct interaction with the user, with two exceptions: Firstly, the user needs to tell the system when a cooking session is finished, such that the recording can be stopped and unused ingredients can be removed from the list. Secondly, the user can manually specify so-called breakpoints before or after important steps in the preparation process. If such a breakpoint is reached during playback, the system compares the states of the kitchen environment at recording time with the state of the kitchen environment at playback time. If there is any difference between these states, the system can react on this unexpected situation. More information on this feature is provided in the following subsection.

Both functions can be invoked via the tangible *control cube* interface that was introduced earlier. In order to do so, the user has to turn the control cube such that the associated side of the cube faces upward.

9.2.2 Instruction Mode

In *instruction mode*, the *Semantic Cookbook* actively supports the user in repeating an existing recipe from the user's own or some other user's collection. The user might either explicitly choose the intended recipe from a list of available recipes, or alternatively just starts with the preparation of a recipe she (partly) remembers while letting the *Semantic Cookbook* try to figure out which recipe is prepared through the use of plan recognition (see subsection on plan recognition below).

Once the correct recipe is known, the *Semantic Cookbook* displays the support screen shown in Figure 9.6 on the kitchen's display. Although the screen looks similar to the one presented in Figure 9.5, its components and behavior differs slightly from observation mode:

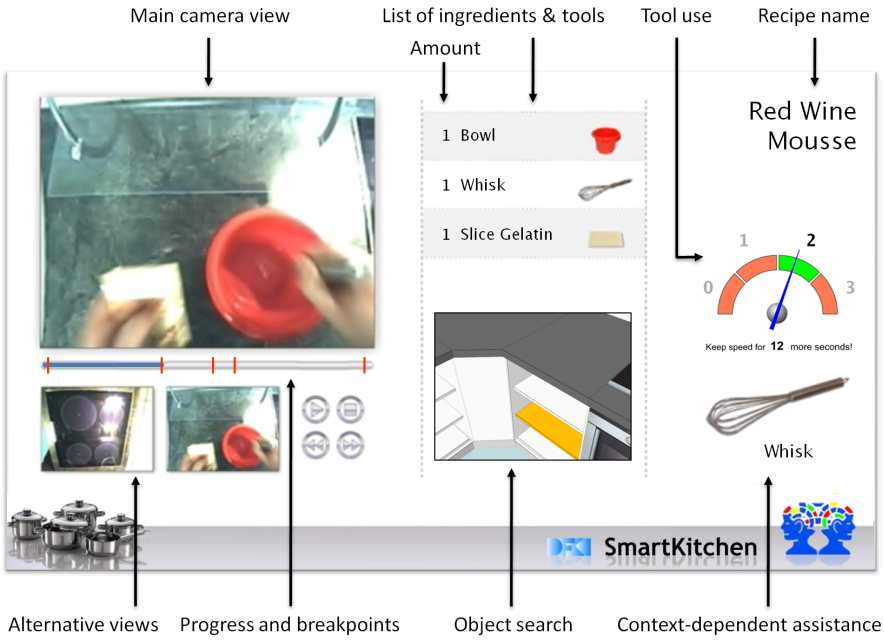


Figure 9.6: Main screen of the *Semantic Cookbook* application while repeating a "Red Wine Mousse" recipe in *instruction mode*.

- **Video Panel**: The multi-angle video which was recorded during the observation phase is replayed. Like in observation mode, the main perspective is chosen by a motion detection algorithm, but the user can override this selection at any point in time by selecting another source from the smaller preview windows. The *progress bar* between the big video window and the preview windows shows the progress of the cooking process. Breakpoints set in observation mode are displayed as vertical red lines on the progress bar. Like in a media player, the user can drag the handle on the progress bar to control the playback of the video and the associated provision of support, or alternatively can use the fast-forward and fast-backward buttons to quickly jump between breakpoints. A user typically moves backwards in the recording to repeat complex or missed preparation instructions. A user might want to skip explanations and hence move forward in the recording if she

already has achived intermediate results of the recipe, for instance by using instant ingredients or by reusing components prepared earlier (like using noodles left over from the day before).

- **List of Ingredients and Tools**: In instruction mode the list of ingredients and tools is populated with all the objects that have been used during the initial preparation of the recipe. The list is ordered according to the objects' time of use, which is inferred from the timestamps of the semantic annotations. The list serves three purposes: (1) It can be used as a shopping list for the chosen recipe. (2) It highlights the objects used during the preparation of a recipe by a graphical animation. This allows the user to easily perceive the objects required now or in the near future, and frees her from the need of constantly following the video. (3) Finally, ingredients and tools already used at replay time are detected via RFID and subsequently grayed out. Thus, user and system can easily check for missed ingredients or forgotten preparation steps. Whenever a breakpoint is reached during playback, the *Semantic Cookbook* checks if all entries in the list up to now have been marked this way. If not, the playback of the presentation is stopped. The playback automatically continues if the problem is solved or the user explicitly commands the system to go on via the control cube. That way, the playback of the recorded recipe is synchronized with the user's actions, which ensures that instructions are given aligned to the user's individual progress.

- **Context-Dependent Assistance**: In instruction mode, the *Semantic Cookbook* provides object-related information similar to the observation mode. In addition, two further kinds of support are provided here: RFID antennas can not only locate objects at the countertop, but also at other places like shelves. The resulting location information is used by the cookbook application to implement a search function for physical objects. The user can trigger the search function by selecting a missing object from the list of required ingredients and tools, or by invoking the search function on the control cube (in this case the next missing object is located). The user then is presented a short movie clip that shows an animated pan shot that starts at the displays location and ends at the selected object's current location. The displayed movie is generated from a 3-D model of the kitchen that was created with the YAMAMOTO system [SH06] (see subsection 8.3.1). If additional data was collected during the use of an object, this information is exploited to assist the user in repeating this action. An example is

the use of the instrumented whisk. Recall, that acceleration sensors attached to the whisk measure the intensity of use. When preparing a recipe for a second time, the actual intensity can be compared with the recorded intensity. Figure 9.6 shows how a tachometer metaphor is used to visualize the associated data.

9.2.3 Application of Plan Recognition

Recipes can be understood as plans that have the goal of preparing certain kinds of food. Accordingly the sensors in the kitchen can be used to observe the user's cooking actions to infer which recipe she most likely intends to prepare. The plan library thereby is given by the set of previously recorded own or shared foreign recipes[1]. Based on the plan recognition results, the *Semantic Cookbook* provides different kinds of support. Figure 9.7 shows the screen that is displayed if the cookbook application neither is in observation mode nor a particular recipe has been selected by the user so far.

- **Prediction of Recipes**: If the user starts cooking without explicitly choosing a recipe, the system tries to infer the intended recipe. Based on the observations made so far it calculates the likelihood of each recipe and presents to the user a sorted list of candidates recipes (see Figure 9.7, top part). Recipes are grouped by categories, and the summarized probability of each group is given by the bar display behind the group title (a completely filled bar denotes probability 1, while a completely empty bar denotes probability 0). The probabilities of individual recipes are given by smaller bars behind the recipes' names. As the user continues with her preparations, the probabilities are updated and the list is reordered.

- **Prediction of Next Preparation Steps**: The system tries to predict the user's probable next actions in order to remind her of upcoming steps and to proactively provide additional instruction on how to use certain tools or ingredients. The top-five list of expected tools or ingredients is shown at the lower left of the plan recognition screen (see Figure 9.8, top part). Each predicted action is represented by an icon, which is shown above the involved object's label and the predicted probability of occurrence.

[1] As the user can use the *Semantic Cookbook*'s observation mode to record new recipes at any point in time, the plan library over time "learns" the user's recipes. Similarly, the probabilities of individual recipes can be learned by observing how often a recipe is prepared in instruction mode.

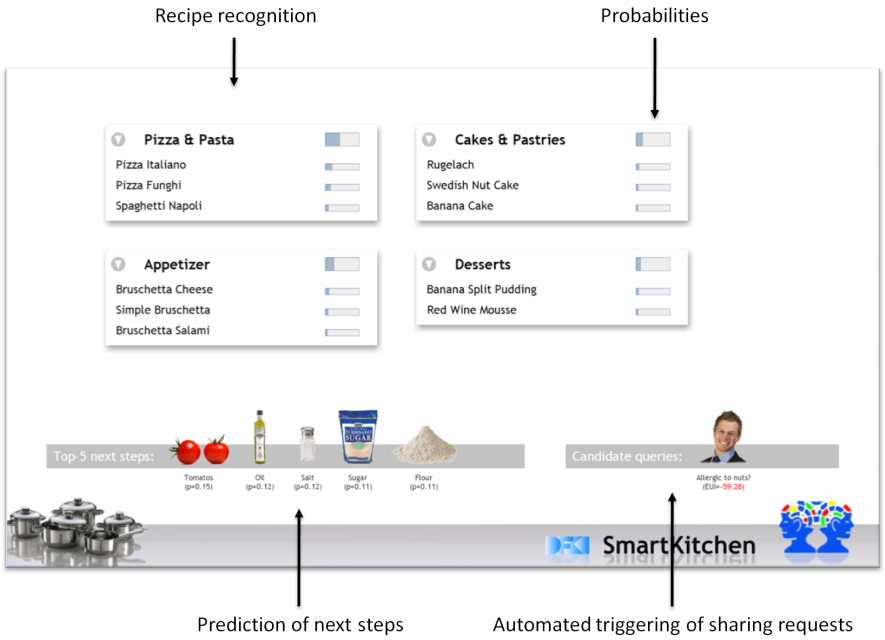


Figure 9.7: Plan recognition screen of the *Semantic Cookbook* application showing different kinds of plan-based support.

- **Automated Triggering of Sharing Requests**: Recall that the *Semantic Cookbook* can query other user's profiles via SharedLife, e.g. in order to learn about their food preferences or allergies. Such information can be exploited by the *Semantic Cookbook* to warn the user in case she intends to prepare something that is unsuitable for expected guests. The two images at the bottom row of Figure 9.8 show an example in which Tim is expected as a guest.

  So far, the system does not know if Tim is allergic to nuts or not. The system could find this out by issuing a sharing request to Tim's SharedLife system which asks for Tim's allergy profile. However, as this is private information, Tim generally has to authorize such request. In order not to annoy Tim with unnecessary request, the system does not issue such requests in advance, but only if the system is sure (to a reasonable extend) that the requested information becomes relevant in the near future. Plan recognition aids the cookbook application with deciding whether to issue a sharing request or not: The information on a potential allergy to nuts is only important, if the plan hypothesis indicates that the preparation of recipe containing nuts is likely.

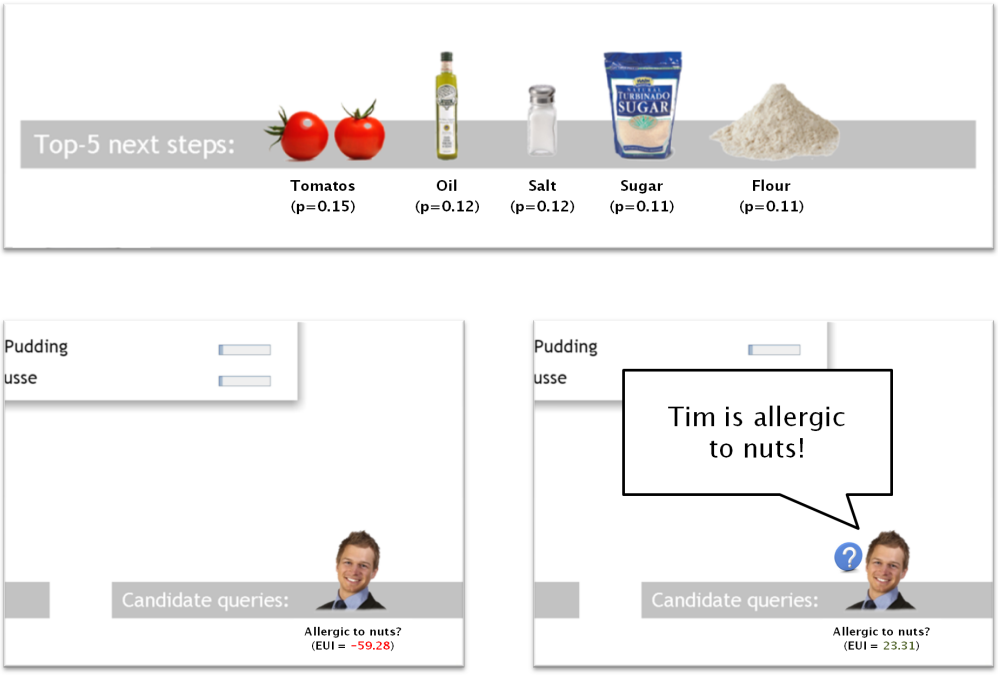


Figure 9.8: Two examples of the use of plan recognition information in the *Semantic Cookbook* application: Prediction of next steps (top) and automated triggering of SharedLife sharing requests based on decision-theoretic utility of information (bottom).

The example of deciding whether to automatically trigger a sharing request or not is related to the problem of sensor selection (see section 2.4), if Tim's SharedLife system is understood as a special kind of sensor. In this case, the utility model for observation information that is presented in chapter 7 can be used to assess the expected utility of information (EUI), which summarizes the expected costs and rewards of some information item. In the example given in the bottom row of Figure 9.8, the resulting EUI value for the current plan hypothesis is displayed below the query label. Initially, the cost of information exceeds the benefit, thus EUI is negative (see Figure 9.8, bottom left) and accordingly the sharing request is not issued. As more observations are made and the resulting hypothesis is refined, the preparation of a recipe containing nuts is assumed to become more likely (this of course depends on the actions taken by the user). In this case, the benefit at some point exceeds the cost, which results in a positive EUI. The sharing request is issued and the response finally is displayed (see Figure 9.8, bottom right).

9.2.4 Extension of Plan Recognition Support

In this section we give further examples of services that can be implemented based on plan recognition in the *Smart Kitchen* environment. The actual implementation of these services lies outside the focus of this thesis and is recommended for future work. Three general areas exist in which the scenario can be extended: Sensors, actuators, and recognized plans. In the following we give examples for each of these areas.

Additional Sensors

In the previous sections we have seen different examples of sensors that can observe the actions of a user in a kitchen environment. By extending the range of observable actions through additional (kinds of) sensors, we can extend the classes of plans that can be recognized in the kitchen environment.

For instance networks of wired and wireless sensors are well suited to observe the interactions of the user with kitchen utensils like our instrumented whisk example that we presented above. Kranz and his colleagues use sensors in a knife and a cutting board (see Figure 9.9.a/b) for detecting context information in the kitchen [KSM+07]. The cutting board is place onto load cells that can detect and distinguish between cutting and chopping actions. The knife is equipped with a force/torque transducer. Based on the signature of the forces that are applied on the knife and the cutting board, Kranz and his colleagues can detect what the user is cutting and how much of it.

A variety of different sensors is applied by Olivier et al. in their *Ambient Kitchen* environment [OXMH09]. Integrated sensors include object mounted accelerometers attached for instance to cooking spoons and spatulas, RFID-tagged paper cookbooks to observe which recipes the user is studying, and a combination of wired and wireless under-floor pressure sensor that allow tracking the movements of the user in the kitchen.

Tapia, Intille, and Larson apply large amounts of very simple state change sensors (Figure 9.9.c) in a home environment for activity recognition [TIL04]. Their sensors measure events like light changes, door-operation (via reed contacts), or pressure mats and switches in chairs or in the bed (Figure 9.9.d). Activity information is inferred by fusing state information from multiple sensors. In an kitchen environment, reasonable state changes that are interesting to observe by such sensors besides light and door operation are the use of electrical appliances, tap use, room temperature, noise level, use of waste bin, or window opening.

Figure 9.9: Additional sensors in the kitchen: Instrumented knife (a), cutting board with load sensors (b), and cheap state change sensors (c) at various places in the home (d).

Additional Actuators

Actuators provide the interface through which support applications assist the user, either directly through the manipulation of the physical environment, or indirectly by the provision of information which helps the user in executing her plans. By extending the set of actuators in the environment, new kinds of plan-recognition-based support services can be realized.

Often information is presented to the user via displays. In the *Smart Kitchen* environment for this purpose a display is mounted over the counter top. While this protects the display from getting dirty during the preparation of food, this setup causes certain inconveniences for the user. At first she might not always look at the display and thus miss important or useful information. Even if information is noticed, she most probably has to change the orientation of her head to perceive the presented orientation, which withdraws her attention from her executed plan.

Better alternatives are displays which are embedded in the kitchen environment. Olivier et al. use projectors to display information at walls near the counter top or even on the counter top itself (see Figure 9.10.a).

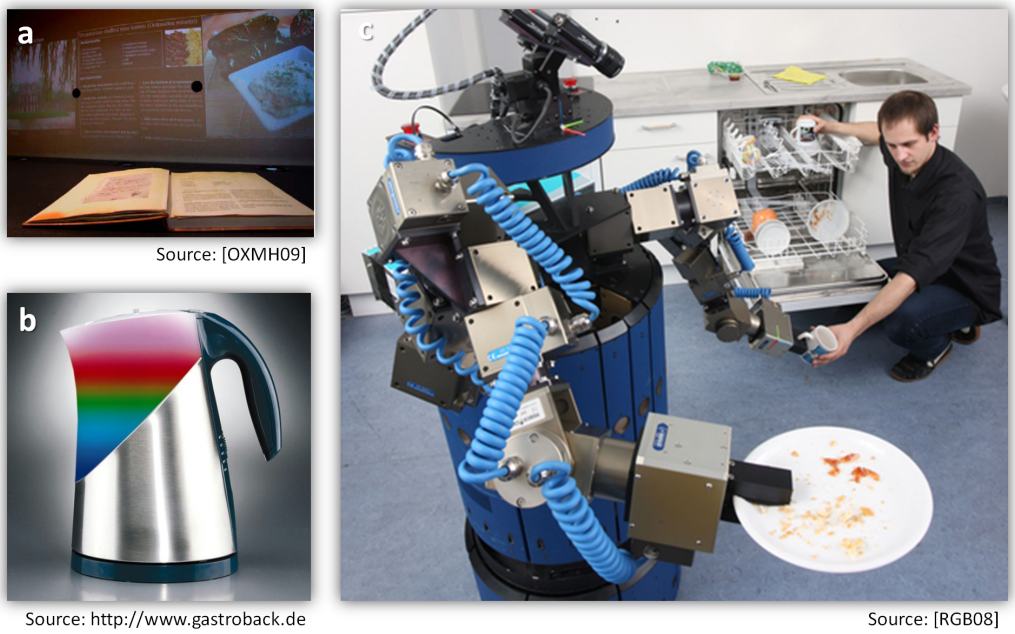


Figure 9.10: Actuators can passively support the user through information display through projection (a) or ambient displays (b), or directly through physical manipulation of the environment, e.g. through a robot (c).

An even more unobtrusive alternative are *ambient displays*, which seamlessly integrate with the environment and can be perceived by the user without focusing her attention. An example from the cooking domain is an electronic kettle which changes the color of its housing depending on the temperature of the contained water (Figure 9.10.b). Such a display could be used by a plan recognition system to "calmly" notify the user on the correct water temperature for his purpose, e.g. depending on the kind of tea the user intends to prepare.

Practical and sophisticated assistance can be provided by systems which can physically manipulate the environment and hence can execute real-world actions on behalf of the user. Rusu et al. propose to use robots for this purpose [RGB08]. They developed a service robot which learns new plans by demonstration and then autonomously acts in an instrumented kitchen environment. If equipped with a plan recognition system, the robot could infer the intentions of its owner and thus could proactively support her with her daily tasks.

Extension of Plan Library

While sensors and actuators provide the technical infrastructure for the provision of proactive support services, the "intelligence" of such services comes from the knowledge about the user's needs, wishes, and intentions. One source of such information is plan recognition, and the foundation of a plan recognition system (one could say its knowledge base) is its plan library. The more plans this plan library contains and the more detailed these plans can describe the user's goals, the better the support is which can be provided on the basis of the plan recognition results. The extension of the quantity of plans and detail of representation in the plan library hence extends the power of the plan recognition system.

In the cooking domain, the most obvious kind of plans to add are recipes. This might not only include completely new recipes, but also variants or variations of known recipes. With such an extension, the system could help the user fixing mishaps and general preparation errors, or encourage her to be creative or to try out new recipes.

On a higher level, plans might consider the preparation of food in a greater context. A smart kitchen for instance might recognize whether the use adheres to some sort of dietary plan. The kitchen then might support the user in preparing varied meals that conform to the dietary plan. A related area is ensuring the compliance of a medical treatment. Here, often constraints between the taking of medicine and food exist, e.g. in the form that certain combinations of food and medicine are not allowed (like milk and antibiotics), or in the form of time constraints between the taking of medicine and eating. A plan-recognition-enabled kitchen could remind the user on her medicine, and propose an order in which to take medicine and food. Plans related to diets and medical compliance are especially interesting in the context of ambient assisted living (AAL) applications.

At last, a smart kitchen could not only recognize and support plans related to the preparation and eating of food, but also to other plans that are typically executed in a kitchen environment. Often the kitchen is a place for social events, like cooking with friends, chatting, or performing hobbies. Whenever such plans are recognized and suitable support is available, a kitchen can support these activities. An example is the adaptation of the environment to the current task (light, music, etc.), or the automated preparation of coffee, or the proposal of common recipes that all friends like, or even new recipes that nobody has prepared before.

9.3 Summary

In this section we presented an instrumented kitchen environment that allows for the observation of its user's behavior via several sensors. User support is provided by the *Semantic Cookbook* application, which assists the user in the preparation of recipes. The cookbook application uses our RePReTo system to infer the recipe that is intended by the user via plan recognition.

We described the technical setup of the environment and the use of our *Semantic Cookbook* application. One goal of the presented environment is to serve as a testbed for the application of plan recognition techniques. As we explained in the previous section, the capabilities of the smart kitchen environment can be further improved through the addition of more sensors and actuators, and through the extension of the plan library to include more plans and plan variants.

We've all heard that a million monkeys banging on a million typewriters will eventually reproduce the entire works of Shakespeare. Now, thanks to the Internet, we know this is not true.

Robert Wilensky

10 Evaluation

One of the main contributions of this thesis is a utility model for observation information in plan recognition applications (see chapter 7), which is motivated by the need to perform sensor selection in real-world environments (see section 2.4). Although the proposed model is derived from theoretical models with strong underlying formal rationale, its *practical performance* to the problem of sensor selection in plan recognition applications still has to be evaluated. In this chapter, we present an *empirical evaluation study*, which has the goal of investigating the following two aspects:

1. **Validity:** The question of the model's validity addresses its general applicability to the problem of sensor selection in plan recognition applications. Here, we test whether the utility model allows for a reliable identification of relevant sensors.

2. **Performance:** The question of the model's performance addresses the absolute quality of the model's estimations regarding sensor utility. Here, we test how useful the information of selected sensors is compared to the optimal case of complete information.

Section 10.1 introduces the evaluation *method* of our study. It motivates the choices that have been made regarding its design and shortly introduces the considered test conditions and assumed knowledge models. Section 10.2 presents the *results* of the performed evaluation trials and gives a short summary of the obtained data. Finally, section 10.3 comprises a *discussion* of the evaluation results. We explain, which conclusions can be drawn with respect to the general performance of the proposed utility model, and discuss the implications of these conclusions on practical applications that want to utilize the proposed utility model for the purpose of sensor selection in real-world plan recognition applications.

10.1 Method

In this section we describe the setup of our evaluation study. First we describe how to evaluate the recognition performance of the proposed state-aware plan recognition approach with respect to intrinsically motivated plan recognition applications (see subsection 10.1.1). Next we introduce a numerical measure for the performance of general sensor selection strategies in extrinsically motivated plan recognition applications (see subsection 10.1.2). This measure provides the primary source of empirical evidence in the following experiments. We then introduce an experiment which uses the presented performance measure to test the utility model's *validity* (see subsection 10.1.3). Next we derive a normalized performance measure, which allows to judge the absolute *performance* of the proposed utility model (see subsection 10.1.4). We conclude this section with a description of the concrete *implementation* of the finally performed evaluation (see subsection 10.1.5).

10.1.1 Performance of State-Aware Plan Recognition

Our evaluation starts by investigating the *recognition performance* of the *state-aware plan recognition approach* that we proposed in chapter 6. This is realized by observing an agent executing a large number of plans from a known plan library, and for each resulting plan hypothesis recording the average *probability* that is assigned to the correct candidate explanation by the plan recognition system. In addition we record the *rank* of the correct candidate explanation, which is the number of explanations to which the plan recognition system assigns a probability that is equal to or higher than the probability assigned to the correct explanation.

As we are interested in the raw plan-recognition performance, we consider plan execution without the assistance of a higher-order support system. The experiment is repeated for different amounts of active sensors in order to understand the influence of limited sensing resources on the recognition performance of the proposed state-aware plan recognition model.

It is important to note that the raw recognition performance of a plan recognition model is only a meaningful performance measure in the case of *extrinsically motivated plan recognition*, where the primary objective is to *maximize the overall recognition rate*. Recall from subsection 7.1.1 that in practical plan recognition applications the alternative case of *extrinsically motivated plan recognition* is generally more relevant. As the primary goal in this case is to maximize the utility realized by the combination of plan recognizer and higher-order support system, a plan recognition model with a lower average recognition performance might under certain circumstances outper-

form a "better" model in terms of outcome utility (see thought experiment in subsection 7.1.1).

Due to the greater practical relevance this thesis in general and the proposed utility model for observation information in particular focus on the case of extrinsically motivated plan recognition. However, the basic state-aware plan-recognition approach can be used in intrinsically motivated plan recognition applications as well. Hence we include an evaluation of the plan recognition model's recognition performance for the sake of completeness.

10.1.2 Utility Model Performance Measure

In order to empirically evaluate the validity and performance of the proposed utility model for the purpose of sensor selection we first introduce a numerical measure of the performance of general sensor selection strategies in plan recognition applications.

Recall from section 7.1 that this thesis focuses on the case of extrinsically motivated plan recognition, where the primary objective is to maximize the outcome utility of the agent's plan execution process through the provision of assistance by the overall system. Among other factors that are mentioned below, the concrete amount of outcome utility depends on the performance of the applied sensor selection strategy respectively its success in selecting "relevant" observation sources.

Figure 10.1 illustrates the existing causal dependencies: A good sensor selection strategy leads to the selection of relevant sensors, which provide meaningful information to the plan recognition system and thus increase the accuracy of the inferred plan hypotheses. This in turn allows the support system to provide more suitable support, which eventually increases the resulting outcome utility.

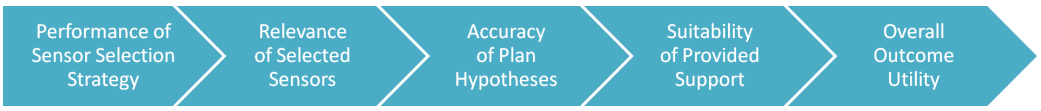


Figure 10.1: The system's outcome utility depends on the utility model's correctness and performance, thus outcome utility can be used as an indirect performance measure.

According to the theoretical models presented in chapters 6 and 7 the stochastic process of supported plan execution under constrained sensing resources (and thus the resulting outcome utility) is completely determined by the following five variables:

1. The agent's plan library
2. The system's support model
3. The cost-reward model
4. The limit of available sensing resources
5. The applied sensor selection strategy

If we can control variables 1–4, then outcome utility provides an indirect measure of the sensor selection strategy's performance. We simply have to observe the agent while it is executing plans with support through the system and then record the true costs of the agent's and system's actions, as well as the rewards of the attained goals as given by the assumed cost-reward model.

We can use this approach to compare the relative performance of two sensor selection strategies A and B for a given plan library, support model, cost-reward model, and sensing resource limit as follows:

- **Preparation:** We choose a multiset Q of test plans from the given plan library, such that the frequencies of plans in Q are distributed according to the probabilities specified in the assumed plan library. In order to create such a set Q of arbitrary size we can understand the plan library as an extended probabilistic automaton (see section 6.1) and then use this automaton as a generator for Q.

- **Condition A:** We let an agent execute all plans from Q, while selecting sensors according to strategy A and providing support based on the resulting plan hypothesis and the given support model. We observe the outcome utilities of all executed plans according to the given cost-reward model and compute their average as $\overline{U}_A$.

- **Condition B:** We repeat the process performed in the previous step, but this time select sensors according to strategy B. Again we compute the average outcome utility as $\overline{U}_B$.

Now, we can compare the relative performance of strategies A and B by considering the two values $\overline{U}_A$ and $\overline{U}_B$. As the only difference between test conditions A and B is the applied sensor selection strategy, a significant difference in average outcome utilities $\overline{U}_A$ and $\overline{U}_B$ indicates a superior performance of strategy A over B (shortly $A > B$) if $\overline{U}_A > \overline{U}_B$ respectively a superior performance of strategy B over A (shortly $A < B$) if $\overline{U}_A < \overline{U}_B$.

10.1.3 Validity of the Utility Model

In this section we describe the experiment that we conducted to test the general validity of the proposed utility model for the purpose of utility-based sensor selection (see section 2.4) in plan recognition applications. A utility model is valid for the purpose of sensor selection when it allows to derive a sensor selection strategy that on average outperforms an uninformed sensor selection strategy. In order to compare the relative performance of an informed and uninformed sensor selection strategy we use the approach presented in the previous subsection 10.1.2. Two relevant outcomes of this comparison have to be distinguished:

- **UNINFORMED $\geq$ INFORMED:** An informed sensor selection strategy might fail to outperform an uninformed sensor selection strategy for one or both of two major reasons: At first, the utility model that guides the search for a good subset of sensors might be invalid, such that high utility ratings do not correspond to useful sensors. At second, the search algorithm itself might be inefficient or erroneous, and thus might fail to find a good subset of sensors. In general, an outcome of UNINFORMED $\geq$ INFORMED thus does not allow us to make any assumption on the validity of our utility model (as long as we do not have proved the correctness of our search algorithm).

- **UNINFORMED $<$ INFORMED:** If our experiment reveals that the informed strategy is superior to the uninformed strategy, then this has to be due to the additional information that the informed strategy has available through the utility model (as long as the informed strategy uses no domain knowledge except the considered utility model). Note that in the case of UNINFORMED $<$ INFORMED the absolute performance and general correctness of the applied search strategy is irrelevant, as this "at worst" only negatively affects the performance of an informed sensor selection strategy, and thus can never result in a false outcome of UNINFORMED $<$ INFORMED. The opposite case can only happen by chance, which we can detect via a significance test.

As uninformed sensor selection strategy we use a strategy RAND that selects sensors at random until the given resource limit is exhausted. Strategy RAND is described in more detail below. As an informed sensor selection strategy that makes use of the proposed utility model we use strategy DT-BNB as introduced in subsection 7.4.2. As DT-BNB is based on a generic branch and bound search, it obviously does not utilize any further domain

knowledge except our utility model, which allows to conclude that our utility model is valid if we observe that $\overline{U}_{RAND} < \overline{U}_{DT-BNB}$.

From the above case distinction we derive the hypotheses for our experiment: As null hypothesis we assume that the average outcome utility under condition DT-BNB does not exceed the average outcome utility under condition RAND:

$$H_0 : \overline{U}_{RAND} \geq \overline{U}_{DT-BNB}$$

The alternative hypothesis of our experiment is that the average outcome utility under condition DT-BNB exceeds the average outcome utility of condition RAND, and thus the proposed utility model is valid:

$$H_1 : \overline{U}_{RAND} < \overline{U}_{DT-BNB}$$

As discussed above the wrong choice or implementation of the informed sensor selection strategy might introduce type II errors (incorrectly accepting H_0), but cannot introduce type I errors (incorrectly rejecting H_0). Thus, our experiment setup is rather conservative, and accordingly the rejection of the null hypothesis is a very strong indicator for the utility model's validity.

Sensor Selection Strategy RAND

Sensor selection strategy RAND represents the case of uninformed sensor selection in our experiments. "Uninformed" in this context means, that the sensor selection strategy does not have access to any domain or background knowledge that might support its decision process. Without such knowledge, a sensor selection strategy at best can randomly "guess" the optimal set of sensors. Strategy RAND starts with an empty set and in each iteration adds a randomly selected sensor which does not exceed the overall resource limit. This process continues until no further sensors can be added.

The exact behavior of uninformed sensor selection strategy RAND is described by Algorithm 5, where S is the set of all sensors, $\hat{r}$ is the given upper resource limit, r is a function which returns the resource consumption of each sensor from S, and $rand(C)$ is a function which randomly chooses one element from a set C.

Algorithm 5: Uninformed sensor selection strategy RAND

select $_{RAND}(S, \hat{r})$:
Data: S: Set of all sensors; $\hat{r}$: Upper resource limit
Result: Optimal set of sensors to activate
begin
  $S' = \emptyset$
  $r' = 0$
  $C = \{s \in S : r(s) \leq \hat{r}\}$
  while $C \neq \emptyset$ **do**
    $s' = rand(C)$
    $S' = S' \cup \{s'\}$
    $r' = r' + r(s')$
    $C = \{s \in S \setminus S' : r(s) \leq \hat{r} - r'\}$
  end
  return S'
end

10.1.4 Performance of the Utility Model

The experiment that we presented in the previous section allows us to learn about the general validity of the proposed utility model for the purpose of sensor selection. In order to learn about its performance (the quality of the resulting set of selected sensors) two general options exist:

- **Relative Performance:** We already described in subsection 10.1.2 how to compare the relative performance of two sensor selection strategies. This allows us to compare strategies like DT-BNB and DT-GREEDY (see section 7.4) to other existing strategies and utility models. To the best of our knowledge there exists no prior work in the domain of sensor selection for plan recognition, thus we are not aware of any other existing sensor selection approach that could be used as a benchmark for our performance evaluation.

- **Absolute Performance:** The second option is to use a measure which allows judging the absolute performance of a utility model respectively derived sensor selection strategies in terms of a meaningful standardized quantity (like realized percentage of maximum outcome value). This allows to rate the absolute performance of single sensor selection strategies without using other strategies as reference points. As a positive side effect, such an absolute measure can also be used to judge the relative performance of different strategies.

As motivated above we are not aware of any other sensor selection strategy in plan recognition applications that may be used as a benchmark for our performance evaluation. Thus, we focus in the following on finding a measure for the absolute performance of a utility model respectively a sensor selection strategy.

While average outcome utility as proposed in subsection 10.1.2 is a good measure to compare the relative performance of two (or more) sensor selection strategies given the same underlying knowledge models, the absolute average outcome depends on the assumed cost-reward model and thus (without any further reference point) is not a meaningful general measure for the absolute performance of a sensor selection strategy respectively utility model. In order to judge (and compare) the absolute performance of utility models and sensor selection strategies independently of the applied cost-reward model, we introduce the *effectiveness factor* measure E of a sensor selection strategy as a normalized utility-based measure for the added value that can be realized by a system that applies a particular sensor selection strategy.

The effectiveness factor normalizes the resulting outcome utility to the interval $[0, 1]$. An effectiveness factor of 0 corresponds to the average absolute utility that is realized by the overall system if *no sensor* is activated (this utility value is denoted $\overline{U}_0$). An effectiveness factor of 1 corresponds to the average absolute utility that is realized by the overall system if *all sensors* are activated (this utility value is denoted $\overline{U}_1$). From the decision-theoretic principle that additional information can never reduce the expected outcome utility (see subsection 7.4.2) we can conclude that the true outcome utility realized with partially activated sensors always lies within the interval $[\overline{U}_0, \overline{U}_1]$. For a concrete average utility value $\overline{U}$, the effectiveness factor E is computed via the following expression:

$$E = \frac{\overline{U} - \overline{U}_0}{\overline{U}_1 - \overline{U}_0}$$

The effectiveness factor can be interpreted as follows: A sensor selection strategy with an effectiveness factor of $E = 0.8$ allows the system to realize on average 80% of the added-value that the system would have been able to generate with full information available. A useful property of the proposed efficiency factor E is that the values of U_0 and U_1 (which unambiguously determine the normalization) only depend on the assumed plan library, support model, and cost-reward model, but are independent of the applied sensor selection strategy. Thus, the same normalization parameters can be used for any considered sensor selection strategy, which preserves the desired property of relative comparability of strategies.

10.1.5 Implementation

The finally performed evaluation comprises two trials: In the first trial we evaluate the recognition performance of the state-aware plan recognition model as described in section 10.1.1. As motivated earlier, the trial is performed without the provision of assistance through a support system. This case is represented by a trivial support model that only comprises the obligatory *none* action, which by definition does not influence the agent's behavior.

The two experiments that we presented in sections 10.1.3 and 10.1.4 can be combined to a single, second trial as follows: We use the approach described in subsection 10.1.2 to calculate the average outcome utilities $\overline{U}_{RAND}$, $\overline{U}_{DT-BNB}$, and $\overline{U}_{DT-GREEDY}$ of our sensor selection strategies RAND, DT-BNB, and DT-GREEDY. We then compute the effectiveness factors E_{RAND}, E_{DT-BNB}, and $E_{DT-GREEDY}$ for each sensor selection strategy.

In order to decide on the validity of our utility model , we then compare the values $\overline{U}_{RAND}$ and $\overline{U}_{DT-BNB}$ (see subsection 10.1.3) and use a *one-sided independent two-sample t-test* to ensure the statistical significance of the results. To judge the absolute performance of the proposed sensor selection strategies DT-BNB and DT-GREEDY we then consider the effectiveness factors E_{DT-BNB} and $E_{DT-GREEDY}$ (we also include E_{RAND} as a reference).

All experiments are conducted with the same plan library and support model (if applicable) with the properties shown in Table 10.1. Each plan in the library has multiple variants, and all variants of a plan lead to the same goal state. Each goal state is associated with a distinct positive reward, while agent and system actions are associated with distinct negative costs. Support actions are only effective at particular states, and each action "forces" the agent to switch to continue with a particular plan (and a randomly chosen variant) if executed in the correct state. If executed in any other state, support actions cause costs but otherwise have no further effect. All models are randomly created within the given bounds in order to minimize the chance of accidentally choosing models which prefer the proposed utility measure.

Number of Plan Execution States	1000	
Number of Agent Actions	30	
Number of Plans/Goal States	30	(plus variants)
Length of Plans (in Agent Actions)	10–25	
Number of Support Actions	30	(plus *none* action)

Table 10.1: Properties of the randomly generated plan library and support model that was used for the evaluation.

The used cost-reward model randomly assigns costs between 0 and 30 to agent and system actions. The rewards for completed plans are randomly assigned from an interval between 0 and 3000. The sensor model that we used assumes the presence of 30 independent sensors, where each sensor detects the execution of one distinct agent action. All sensors equally consume one unit of an imaginary resource.

In order to shed light on the performance of the proposed sensor selection strategies under different upper resource bounds $\hat{r}$ we repeat our experiments for $\hat{r} \in \{0, 1, \ldots, 30\}$, which corresponds to a sensor selection rate of $0\%, \frac{1}{30}\%, \ldots, 100\%$.

Due to the probabilistic nature of the agent's plan selection and execution process as well as of the considered sensor selection strategy RAND a huge number of plan executions need to be observed in order to calculate a representative average outcome utility $\overline{U}$. In total, we execute 1000 plans for each of the $3 \cdot 31 = 93$ test conditions (3 sensor selection strategies and 31 different values of $\hat{r}$).

Obviously, such an experiment is impossible to conduct with human users in a physical environment due to the sheer amount of actions and plans that need to be executed by the users. In similar experiments one thus often uses corpora of prerecorded data as input to the test algorithms. This approach also is not applicable to our case, as the evaluation requires a true interaction between the agent and the support system at runtime, as the support system actively influences the agent's behavior. This interaction is impossible to archive with prerecorded data.

We thus choose a setup in which plans are executed by a simulated agent in a simulated environment. During plan execution, the extended plan recognition DBN (see section 7.3) is used by the simulation system to generate matching observation data for the simulated sensors. The internal state of the agent and the environment are only known to the simulation system, and in particular are hidden from the plan recognition component. After each agent action, the considered sensor selection strategy is used to decide on a new set of active sensors. Simulated observation data is only forwarded to active sensors, while observation data of inactive sensors is discarded by the simulation system.

Sensor selection, plan recognition, and provision of support is realized through our proof-of-concept implementation of the RePReTo toolkit that we describe in chapter 8. For each executed plan, the simulation system records the true outcome utility based the executed agent and system actions and the reached goal. The simulation system further collects information about the runtime of the sensor selection algorithm in order to judge the runtime performance of different sensor selection strategies.

10.2 Results

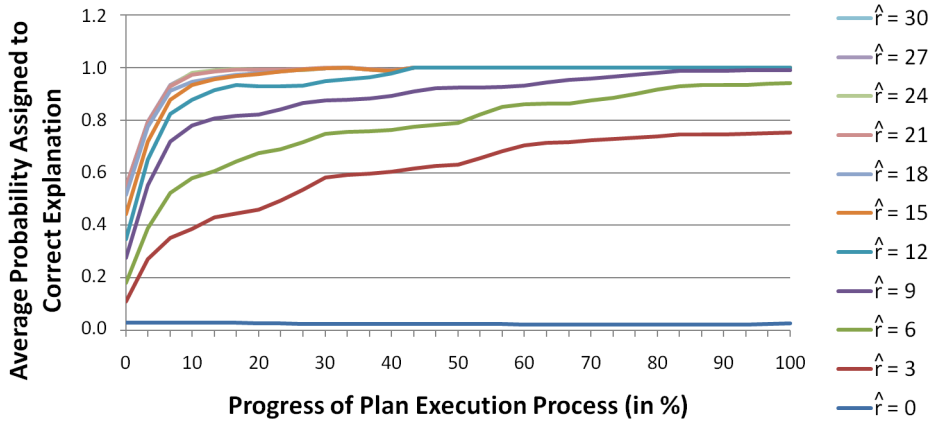
In this section we present the numerical results of the evaluation that we described in the previous section.

10.2.1 Performance of State-Aware Plan Recognition

The first results that we present relate to the recognition performance of the proposed state-aware plan recognition model according to the general idea described in subsection 10.1.1.

At first, we consider the average probability that in each plan recognition iteration is assigned by the state-aware plan recognition model to the correct candidate explanation. Figure 10.2 shows the resulting probability values (y-axis) at different stages of the agent's plan selection and execution process (x-axis), where a point x represents the case that $x\%$ of the steps of the chosen plan have already been executed by the agent (probability values are interpolated if necessary). The resulting probabilities for different degrees of available resources $\hat{r}$ respectively amounts of selected sensors are shown by individual curves in Figure 10.2. In this experiment sensor selection strategy DT-BNB was used to choose the set of active sensors in each iteration.

Figure 10.2: Average Probability Assigned to Correct Explanation



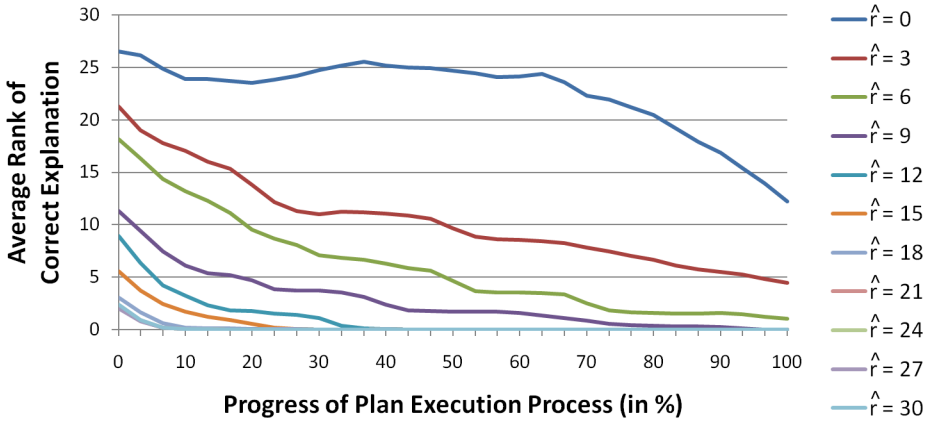
The presented figure shows that with no observation information available (case $\hat{r} = 0$) the plan recognition model is unable to narrow down the correct hypothesis, which results in a constant assignment of the average a-priori probability to the correct explanation. With more sensing resources (and thus more observation information) available, the plan recognition model can

more reliably and earlier identify the correct candidate explanation. With full observation information available (case $\hat{r} = 30$) the plan recognition model on average assigns a probability of nearly 1 to the correct explanation (plan execution state) after the execution of 10% of the actions of a plan.

A few more words have to be said on the probability values for non-extreme resource limits ($0 < \hat{r} < 30$). As we mentioned earlier, raw recognition performance is only a reasonable performance measure in the case of *intrinsically* motivated plan recognition, hence we assumed for this experiment that no support is provided during the execution of plan. Our utility model (and thus the used sensor selection strategy DT-BNB) however is expressed in terms of the utility of support, and hence is only defined in the case of *extrinsically* motivated plan recognition. In order to be able to include the utility values under limited resources as a rough reference, we assumed for our sensor selection strategy the support model that we use in the other experiments that focus on extrinsic utility. Hence, the exact probability values for the cases $0 < \hat{r} < 30$ have to be taken with a grain of salt.

Assigned probability alone is not always a performance measure in intrinsically motivated plan recognition. Especially if the set of possible candidate explanations is large, the correct explanation might be assigned a rather small probability, but might nevertheless be the highest ranked explanation in the candidate set. We thus additionally considered the *average rank* of the correct explanation, which is the number of explanations with a probability equal to or higher than the probability of the correct explanation. Figure 10.3 shows the average rank of the correct explanation (y-axis) at different stages of the plan execution process (x-axis) for different values of $\hat{r}$. For $0 < \hat{r} < 30$ the same considerations as mentioned above do hold here.

Figure 10.3: Average Rank of Correct Explanation

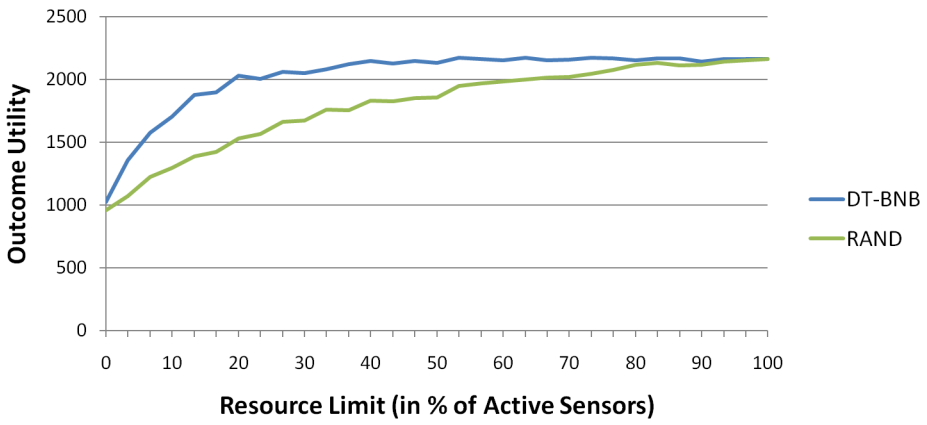


10.2.2 Relative Performance of RAND and DT-BNB

By comparing the performance of sensor selection strategies RAND and DT-BNB we test the validity of the proposed utility model for observation information in plan recognition applications as described in subsection 10.1.3.

The diagram in Figure 10.4 shows the average outcome utilities for both sensor selection strategies at various assumed upper resource limits (respectively ratios of selected sensors). As expected, the average outcome utilities of RAND and DT-BNB equal at the extreme ends (0% and 100%) of the resource limit range, as the choice of the sensor selection strategy is irrelevant if none or all sensors are selected.

Figure 10.4: Relative Performance of DT-BNB Compared to RAND

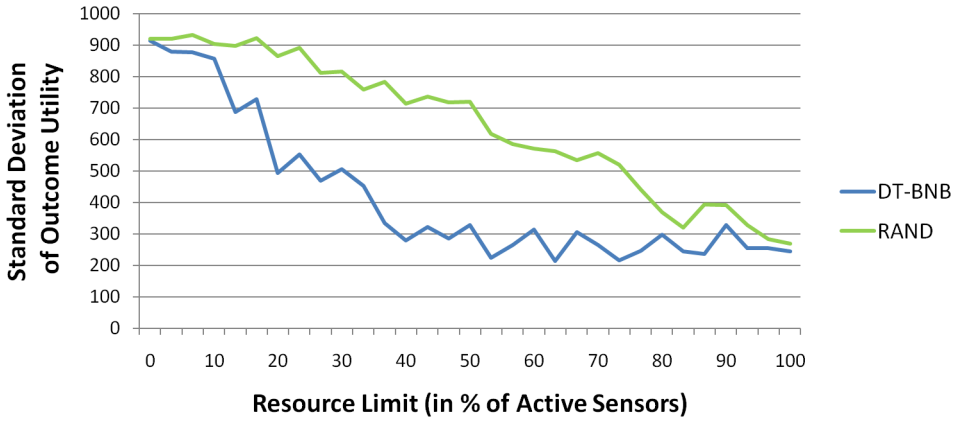


The diagram further indicates that the performance of DT-BNB exceeds the performance of RAND for the considered randomly chosen models in the non-extreme area of the resource limit range. This suggests to reject our null hypothesis H_0, which states that DT-BNB does not outperform RAND. We use a *one-sided independent two-sample t-test* with 1998 degrees of freedom to test whether this finding is statistically relevant. The evaluation of our data shows that at a significance level of 0.99 the null hypothesis is rejected for $\hat{r} \in \{1, \ldots, 26\}$, which approximately equals an interval of $[3.3, 86.6]$ percent of active sensors.

In addition to the performance we also compare the standard deviations of outcome values for strategies DT-BNB and RAND. These numbers allow us to learn about the "stability" of outcome values achieved by both strategies. The results are depicted in Figure 10.5. The standard deviations for strategies DT-BNB and RAND again are virtually equal at the extreme ends of the resource limit range for the same reasons given in the case of outcome

value (see above). In the middle of the resource limit range, the standard deviation of outcome values of DT-BNB is consistently lower than for RAND. This is conform to the intuition that RAND sometimes is lucky guessing relevant sensors and sometimes is not, while DT-BNB achieves more "stable" and thus predictable results in the average case.

Figure 10.5: Deviation of Outcome Values for DT-BNB and RAND

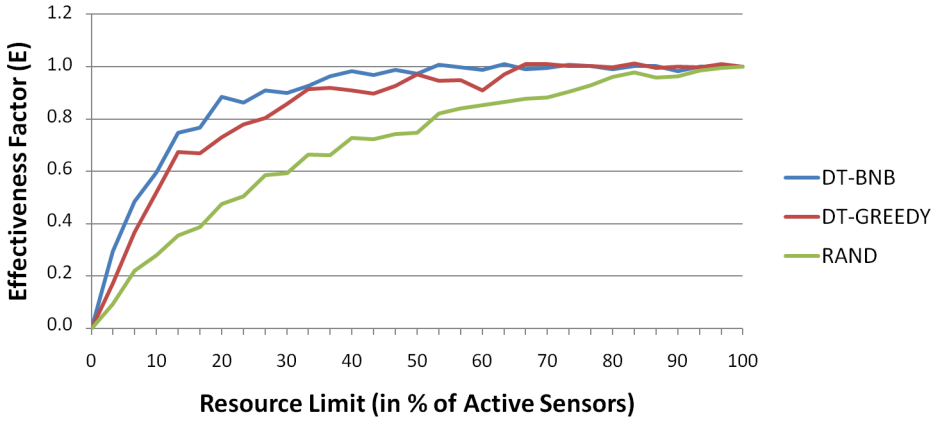


10.2.3 Effectiveness of DT-BNB and DT-GREEDY

In order to learn about the absolute performance of sensor selection strategies DT-BNB and DT-GREEDY we compute and compare their effectiveness factors E_{DT-BNB} and $E_{DT-GREEDY}$ as described in subsection 10.1.4. As a further reference point, we also compute and include the effectiveness factor of the uninformed sensor selection strategy RAND.

The effectiveness factors that result for the three considered strategies and different upper resource limits/ratios of selected sensors are shown in Figure 10.6. Again all strategies perform equally well if none or all sensors are selected. The relative difference between strategies DT-BNB and RAND is not affected by normalization, thus both curves have the same shape and relative distance as shown earlier in Figure 10.4. In addition, we can see that the heuristic decision-theoretic strategy DT-GREEDY is a good approximation of the optimal strategy DT-BNB. It performs slightly inferior to DT-BNB, but still significantly outperforms uninformed sensor selection strategy RAND for $\hat{r} \in \{2, \ldots, 27\}$ at a significance level of 0.99.

Figure 10.6: Effectiveness of DT-BNB, DT-GREEDY, and RAND



10.2.4 Runtime Performance

Next we compared the runtime performance of sensor selection strategies DT-BNB and DT-GREEDY. Strategy RAND again is included as a reference. As explained in subsection 10.1.5, the simulation system for each invocation of the sensor selection component recorded the resulting runtime in milliseconds. All measurements have been taken with our Java-based proof-of-concept implementation of the RePReTo toolkit on a PC with the technical configuration shown in Table 10.2. Timing information was acquired through the Java-wide system clock, which has a resolution of 10 milliseconds.

CPU	Intel E6400 Core 2 Duo, 2.13 MHz
RAM	2 GB
Operating System	Windows XP (SP2)
Virtual Machine	Sun JRE 1.6

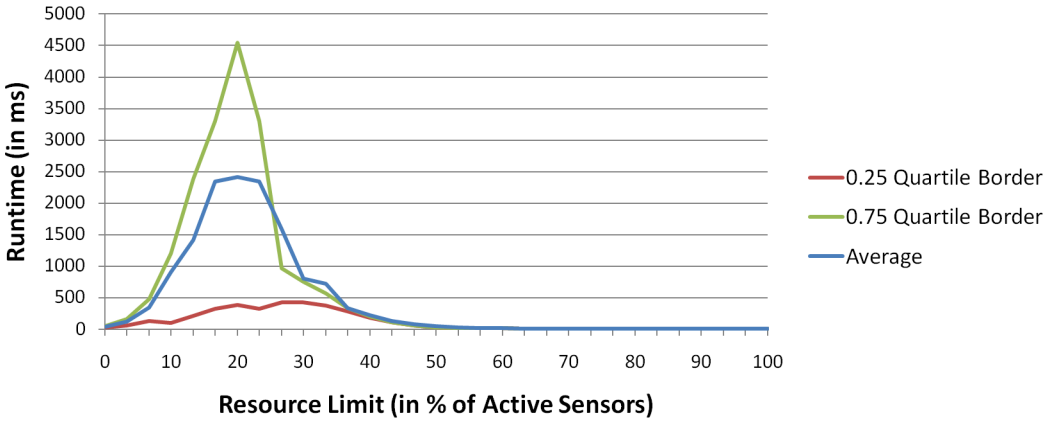
Table 10.2: Technical configuration of the PC used for the evaluation.

One iteration of random sensor selection with algorithm RAND in all cases terminated within less than 10 milliseconds, independent of the given upper resource limit $\hat{r}$. As this is below the resolution of the applied timer, we cannot give the exact runtime here. The same problem exists when measuring the runtime of our decision-theoretic heuristic DT-GREEDY, which terminates under 10 milliseconds in most of the cases. In all other cases, DT-GREEDY terminates in less than 20 milliseconds.

The most interesting data regarding runtime performance was gathered for strategy DT-BNB. As motivated in subsection 7.4.3, the exact number of subsets of sensors that are considered by DT-BNB (and thus the resulting runtime) can usually not be predicted. However, our evaluation reveals that a notable relation between runtime and the chosen upper resource limit exists.

Figure 10.7 shows the average and 0.25/0.75 quartiles runtimes of DT-BNB for different values of $\hat{r}$ (respectively resulting ratios of active sensors). In practical applications not only the average runtime but also the worst-case runtime is relevant. Worst-case runtime is shown in Figure 10.7 (two separate diagrams are given here to avoid scaling issues).

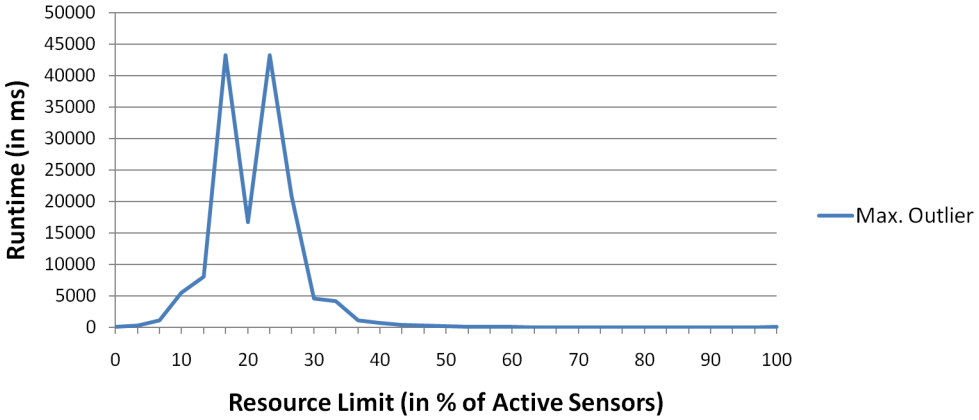
Figure 10.7: Average/Interquartile Runtime of DT-BNB



The numbers reveal two interesting properties of DT-BNB. Although 75% of the sensor selection runs terminated within at last 4500 milliseconds, the largest outlier was at about 44 seconds. This emphasizes the general unpredictability of runtimes of optimal branch and bound approaches. However, it becomes obvious from the figures that average, interquartile, and worst-case runtime performance all have their maximum values in the interval between 10 and 30 percent of active sensors. This corresponds to the fact that the largest performance difference between DT-BNB and RAND (and – although more subtle – also between DT-BNB and DT-GREEDY) can be observed in the same interval between 10 and 30 percent of active sensors (see Figure 10.6).

This observation indicates, that finding an optimal solution to the sensor selection problem in this interval is particular difficult, and thus an optimal strategy has to invest more effort (runtime) to find such a solution. Although

Figure 10.8: Worst-Case Runtime of DT-BNB



a detailed analysis of this assumption goes beyond the scope of this thesis, we share a few thoughts on possible reasons for this notable property at the end of the following discussion of results.

10.3 Summary

The results presented in the previous section provide *significant evidence* that the proposed utility model for observation information in plan recognition applications is *valid* and *applicable to the problem of sensor selection*. The evaluation further shows, that for the assumed models an optimal sensor selection strategy based on the proposed utility model can achieve 90% of the system's performance under full information with only 20% of active sensors (see Figure 10.6). In comparison an uninformed sensor selection strategy requires over 70% of active sensors to achieve the same level of performance.

As the resulting runtime for branch and bound approaches like DT-BNB generally is not predictable, we further presented a heuristic strategy DT-GREEDY. The performance results show, that DT-GREEDY is a *suitable approximation* of DT-BNB which also significantly outperforms RAND. With the proposed heuristic, a performance level of 90% is realized with a ratio of roughly 35% of active sensors.

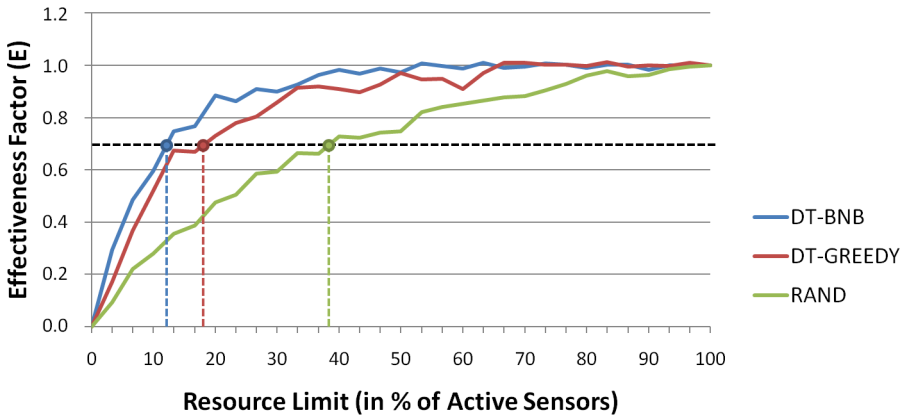
For some exemplary resource bounds Table 10.3 summarizes the rounded effectiveness factors for strategies DT-BNB, DT-GREEDY, and RAND. The findings regarding relative performance are statistically significant for all except extreme resource bounds. The latter aspect is obvious, as sensor selection is pointless when (nearly) none or all sensors are selected.

Active Sensors	$\mathbf{E}_{DT-BNB}$	$\mathbf{E}_{DT-GREEDY}$	$\mathbf{E}_{RAND}$
0%	0.00	0.00	0.00
20%	0.88	0.73	0.47
40%	0.98	0.91	0.73
60%	0.99	0.91	0.85
80%	0.99	0.99	0.96
100%	1.00	1.00	1.00

Table 10.3: Comparison of effectiveness factors for different ratios of active sensors (rounded).

The performed evaluation does not only allow to predict the *average performance level* for a given resource bound, but also to predict the *required amount of resources* for an intended performance level. This for instance is useful if the amount of consumed resources should be limited due to economical reasons, but a certain *minimum performance level* should be maintained. Then, the effectiveness factor diagram known from Figure 10.6 can be used to read from it the required ratio of active sensors for a given minimum performance level and sensor selection strategy. Figure 10.9 gives an example in which the required ratio of active sensors for a minimum performance level of 70% (indicated by the black dashed line) should be identified. From the graph we can read that the required minimum ratios of active sensors are approximately 12% for DT-BNB, 17% for DT-GREEDY, and 38% for RAND.

Figure 10.9: Exemplary Estimation of Required Resources for an Intended Performance Level of 70%



A second focus of our evaluation was on the *runtime performance* of strategies DT-BNB and DT-GREEDY. As expected, the runtime of DT-BNB varied widely and went up to 44 seconds in the worst case. This is not sufficient for most real-time applications. However, our experiments showed that the runtime of our well-performing heuristic DT-GREEDY with a maximum of 20 milliseconds is *suitable for most real-time applications*. This is a very interesting and relevant finding, as it allows to apply the proposed models and approaches in broad range of practical applications.

At last we briefly discuss the peak in runtime that we observed for strategy DT-BNB in the interval between 10% and 30% of active sensors in subsection 10.2.4. We approach an explanation by first considering the opposite cases of very low and very high resource limits. Recall that branch-and-bound optimizes the search by discarding fruitless branches in the search tree. In case of low resource limits, branch and bound can cut off branches early, as the resource limit is exhausted after adding just a few sensors. In case of high resource limits, more observation sources can be selected, which on the one hand allows the sensor selection approach to be less "picky" with spending resources, and on the other hand allows the overall system to better narrow down the plan hypothesis, which increases the utility model's ability to provide tighter upper bounds to the branch and bound algorithm. In the critical interval between 10% and 30% we thus suspect, that there are already enough resources available to yield a large number of candidate sensors sets, while at the same time the resource limit is still so tight that exchanging single sensors might make a big difference in expected utility, and thus a huge number of candidate solutions have to be considered.

An alternative explanation for the observed runtime behavior of DT-BNB might be that the chosen upper bound is too loose in the critical interval, and thus unfruitful branches in the search tree cannot be identified in all cases[1]. A possible reason for this is that Algorithm 3 always uses the maximum candidate solution in a subtree to compute the upper bound of that tree (see subsection 7.4.2). Thereby, it does not check if the maximum solution exceeds the resource limit. This would especially lead to loose bounds in the case of low or mid range resource bounds. If this assumption holds, then the performance of DT-BNB can be improved in the critical interval if we are able to find a better upper bound. However, finding a better bound is not trivial, as it has to be sufficiently simple to compute in order to speed up the search process. We propose to investigate (and if possible improve) the quality of the described bound in the context of future research.

[1]Note that a bound that is too loose negatively affects runtime performance but does *not* affect the correctness or optimality of the found solution.

11 Conclusion

In this chapter we summarize the results of the research underlying this thesis and give an outlook on possible directions for future research on the topic of plan recognition in instrumented environments.

11.1 Research Questions Revisited

In the introduction to this thesis we formulated seven research questions (see section 1.3) which guided the work that we presented in the previous chapters. These questions are now revisited, and for each research question a summary of the obtained results is presented.

1. **Representation of State Information:** *What is a suitable plan selection and execution model that explicitly considers state information?*

   We proposed a plan selection and execution model that is based on the concept of a finite *plan execution state space*, which represents the *abstract state* of the agent's plan selection and execution process. Abstract states model the progress of plan execution, and implicitly represent all aspects that might influence the future evolution of the plan. This includes aspects like the agent's state of mind or the physical state of the environment. The plan execution state space includes an obligatory start state, several intermediate states, and a set of goal states, where each goal state represents the completion of an individual plan. The observed agent executes actions from its action repertoire in order to move from the start state towards a desired goal state via a series of intermediate states. We assumed that abstract plan execution states correspond to characteristic (partial) states of the physical environment, which result from the execution of the previously selected actions. Dependencies between abstract plan execution states and concrete partial states of the physical environment are made explicit by a *sensor model* (see research question 2).

The actual plan selection and execution process is represented by an extended *probabilistic automaton* over the plan execution state space. For each state the automaton defines a probability distribution over possible actions, and for each state and action the automaton defines a probability distribution over possible transitions. From both, we can easily derive the probabilities of individual plans given the current plan execution state. State space, action repertoire, and probabilistic automaton together comprise the agent's *plan library*, which is assumed to correctly and completely describe the agent's behavior.

2. **State-Aware Plan Recognition:** *How can this model be used to perform state-aware plan recognition?*

   Given the state-based plan selection and execution model, plan recognition becomes the problem of estimating the current (hidden) plan execution state based on partial sensor observations. A plan hypothesis then is a probability distribution over the plan execution state space. In order to perform this kind of reasoning, we have to link concrete sensor observations to abstract plan execution states. For this purpose we introduced a *sensor model*, which uses a *Bayesian network* approach to describe the conditional dependencies between executed actions, plan execution state transitions, and the resulting sensor readings.

   To model (and reason about) the evolution of the observed agent's plan selection and execution process over time, we exploit the *Markov property* of the plan library, which states that the action that is selected by the agent as well as the resulting transition (and thus the follow-up plan execution state) only depend on the current plan execution state. Processes with the Markov property can be modeled by *dynamic Bayesian networks* (DBNs). We showed how to derive such a DBN from the plan library. By fusing the resulting DBN with the sensor model we constructed a *combined dynamic Bayesian plan recognition network*, which can be used to perform plan recognition on action observations as well as on state observations. We also explained how this network can be used to predict future actions and intended goal states.

3. **Measure of Utility:** *What is a good measure of the utility of observation information with respect to the plan recognition problem?*

   The immediate benefit of observation information lies in its ability to improve the accuracy of the plan hypothesis. In order to quantify the resulting utility, we considered the assumed source of motivation for plan recognition. Here, we distinguished between intrinsic and extrinsic motivation: In the case of *intrinsic motivation*, plan recognition is

performed with the primary goal of learning about the agent's plans; hence all plans are equally important to recognize. In this case, a high utility value correlates with a high overall *rate of correctly recognized plans*. In the case of *extrinsic motivation*, plan recognition is performed to allow some external support system to provide proactive assistance, hence plans on which the support system can provide significant assistance are more useful to recognize than plans on which no or less important assistance can be provided. In this case, the utility of plan recognition correlates with the *added value* that the support system can realize based on the plan recognition results.

In practical applications plan recognition is mostly performed due to extrinsic motivation, thus we defined our utility measure for observation information as the delta of added value that is expected to be generated by the combined plan recognition and support system with and without having the considered observation information available.

4. **Influencing Factors:** *What are the factors that influence the concrete amount of observation information utility?*

   In the considered case of extrinsically motivated plan recognition, observation information can improve the accuracy of the resulting plan hypothesis, which in turn allows the support system to choose better-suited and thus more valuable support services. The concrete amount of added value which is finally realized by the combined plan recognition and support system then depends on the following three main factors: At first, it depends on the *influence of observation information on the resulting plan hypothesis*. Secondly, it depends on the *influence of the resulting plan hypothesis on the support system's decision making* regarding the support to provide. At last, it depends on the concrete *added value that is realized by the support action* which is finally chosen and executed by the support system.

   The added value of support actions depends on the way these actions influence the observed agent's plan selection and execution process, in particular on the resulting decrease in cost of plan execution and the resulting increase in reward associated with the finally reached goal. These factors for instance can be influenced by the support system through the execution of actions on behalf of the observed agent, or by pointing out higher-valued plan alternatives to the observed agent. The sum of cost savings and increase in reward then determines the added value that is realized by the execution of a support action.

5. **Formal Framework:** *Which formal framework can be used to model the relationship between these factors and the utility of information?*

   The first relevant factor is the influence of observation information on the resulting plan hypothesis. We showed how the *combined plan recognition DBN* can be used to predict this influence for given observation information. The second and third factors are the influence of the plan hypothesis on the chosen support action and the resulting added value. We proposed the use of *decision theory* as formal framework for the representation of these two factors. Decision theory is concerned with *decision making under uncertainty* and the resulting *outcome utilities*, and thus allows for considering both factors in a consistent integrated manner.

   We introduced the *support decision problem*, which is for an external support system to decide on the best support action to execute given the current plan recognition hypothesis. In this context, uncertainty arises from incomplete knowledge about the observed agent's actual plan execution state (and thus its intentions and plans). In order to model the effects of support actions on the observed agent's plan execution process and the resulting added value, we introduced two additional models: The *support model* locally augments the agent's plan library in order to describe the effects of support actions on the observed agent's plan selection and execution process. The *cost-reward model* specifies the costs of agent and system actions, and the rewards associated with the attainment of intermediate and final goals.

6. **Estimation of Utility:** *How can this framework be used to estimate the expected utility of future observation information?*

   In order to compute the maximum expected utility of the best support action given the current plan hypothesis (and thus the extrinsic utility of observation information), we have to *solve the support decision problem*. The support decision problem is a sequential decision problem, which means that the finally resulting outcome utility does not only depend on the decision made in the current time step, but also on decisions that have to be made in future time steps (in our case decisions on support that possibly is provided during the remainder of the executed plan).

   We described how to derive a *partially observable Markov decision process* (POMDP) that represents the support decision problem from the observed agent's plan library, support model, and cost-reward model.

A POMDP models a sequential decision process, in which the underlying state (here the current plan execution state) cannot be directly observed. The constructed POMDP can be solved by using existing (exact or approximate) standard algorithms. The resulting *POMDP policy* finally allows for every plan hypothesis to compute the optimal system action (here support service) and the associated maximum expected utility. Solving a POMDP generally is a time-consuming process. We proposed to perform this process offline and to use the resulting policy at runtime to efficiently decide on the optimal support and to compute the associated values of maximum expected utility.

7. **Performance:** *How does the proposed decision-theoretic utility model perform in sensor selection applications?*

   We evaluated the validity and performance of the proposed utility model in a randomized empirical study. An artificial agent was observed in a simulated environment during the repeated execution of randomly selected plans from a known plan library. Different algorithms were used to select among a given set of virtual sensors. Resulting observation data was passed to a plan recognition system, and the inferred plan hypotheses were provided to a decision-making component, which then decided on the support to provide. For each executed plan the true costs and rewards were recorded. The *effectiveness* of the proposed sensor selection algorithms was computed by comparing the average outcome values for individual ratios of activated sensors with the outcome values in the cases of no observation information (all sensors inactive) and full observation information (all sensors active).

   Evaluation results showed that the average outcome that was realized when sensors are selected based on the proposed utility model is significant higher than in the case of uninformed (random) sensor selection. This allowed us to conclude that the proposed utility model is a valid measure for the utility of observation information in extrinsically motivated plan recognition applications. The evaluation furthermore showed, that runtime performance on a 1000 state plan library is sufficient for real-time applications ($< 20ms$ for one iteration of sensor selection) with an approximate heuristic sensor selection strategy.

11.2 Scientific Contributions

This thesis advances the state-of-the-art of plan recognition in instrumented environments through the following *main contributions*:

- **State-Aware Plan Recognition Approach**

  We proposed a *plan recognition approach* which is based on a formal model of the agent's plan selection and execution process that explicitly takes into account information on the physical state of the environment and the mental state of the agent. The agent's plan selection and execution process is represented as an *extended probabilistic automaton*, from which a *dynamic Bayesian network* (DBN) is derived. The DBN allows for solving the plan recognition problem at runtime. Due to the explicit representation of state information in the formal model, the resulting plan recognition approach allows for the natural consideration of action observations as well as partial state observations. This supports the application of plan recognition in instrumented environments which have to rely on physical sensors to observe the agent.

- **Decision-Theoretic Utility Model for Observation Information**

  We discussed the *nature of observation utility* in plan recognition applications and introduced a distinction between *intrinsic* and *extrinsic* utility. For the case of extrinsic utility we proposed a *decision-theoretic utility model for observation information in plan recognition applications*, which can be used to decide on the best subset of sensors to activate given the current plan hypothesis. The developed utility model defines utility in terms of the added value that is realized by the overall support system. To calculate this added value, we derive a *Partially Observable Markov Decision Process* (POMDP) from the system's *Support Decision Problem*. The POMDP can then be solved offline, and the resulting policy is used to efficiently compute the utility of observation information at runtime. *To the best of our knowledge we are the first to address the problem of generic observation utility and sensor selection in the context of plan recognition applications.*

- **Decision-Theoretic Sensor Selection Strategies**

  We presented two *sensor selection strategies* which make use of the proposed utility model for observation information in plan recognition applications. Strategy DT-BNB uses a *branch and bound* approach to find an *optimal solution* of the sensor selection problem. The proposed utility model is used by DT-BNB to judge the quality of found solutions and to compute upper bounds of candidate sets as required by the generic branch and bound algorithm. The second strategy DT-GREEDY uses a *greedy heuristic* to find an *approximate solution* of the sensor selection problem. DT-GREEDY uses the proposed utility model to decide on the best candidate sensor to pick next.

- **Demonstration of General Feasibility of Sensor Selection**

  We conducted an *empirical evaluation study* which provides very *strong evidence* that the application of sensor selection techniques in resource-constrained plan recognition applications is *generally feasible and useful*, and can *significantly improve* the performance of the overall system. The evaluation study further shows, that the proposed concept of *extrinsic utility* is a *suitable measure* for the estimation of observation information utility, and that the utility model that we developed based on this foundation is *valid* with respect to the problem of sensor selection. Evaluation results further show, that the proposed sensor selection strategies DT-BNB and DT-GREEDY clearly *outperform* an uninformed sensor selection strategy and that the resulting *runtime performance* of the heuristic strategy DT-GREEDY is suitable for a broad range of *real-time applications*.

- **Simulation Approach for Predicting System Performance**

  We applied a simulation approach to predict the overall performance of systems that utilize plan recognition to trigger value added services based on the agent's behavior. This approach allows to compute the average outcome utility for every combination of knowledge models, plan recognition systems, sensor selection strategy, and resource limit. It further allows to compare the performance of two or more variants of some component or knowledge model if all other components and models are kept fixed. This approach was used to *evaluate the performance of sensor selection* in plan recognition applications as described above. In addition, this approach can be used to assist the designer of an environment in *finding the minimal resource bound* required for a given service level, or in *solving the problem of sensor positioning* by allowing different sensor setups (represented by different sensor models) to be compared with respect to the resulting outcome utility of the overall system without the need for installing them in the actual environment. The resulting information can be used to either decide on the best location to position a particular sensor, or even to decide whether a particular sensor is worth to be used at all (which is the case if its average added utility exceeds its transcribed costs).

- **RePReTo System**

  We implemented RePReTo, the *Resource-Aware Plan Recognition Toolkit*, which provides a *generic and reusable framework* for applications that want to utilize plan recognition in order to realize proactive

assistance in instrumented environments. RePReTo implements all required knowledge models and software components, and controls their causal and temporal interrelations. RePReTo provides applications with support for *plan recognition*, *sensor selection for plan recognition*, and *solving of the support decision problem*. RePReTo is implemented as a multi-threaded Java library, which allows existing and future applications to realize proactive support services. In the context of RePReTo we also discussed *methods for the acquisition of knowledge models* that are required for the purpose of plan recognition and sensor selection.

- **Smart Kitchen Test and Demonstration Environment**

  We designed and implemented an instrumented kitchen environment as a practical *demonstration scenario* and *testbed* for the application of plan recognition in real-world application domains. We further implemented the *Semantic Cookbook* application as an example of a *user support application with plan recognition support*. The application supports the user in the preparation of own or foreign recipes in the *Smart Kitchen* environment. The application uses plan recognition to detect the intended recipe and to offer proactive assistance.

11.3 Outlook

As one of its major contributions this thesis pioneered the application of sensor selection in generic plan recognition applications. During our research in this novel field we came across many interesting related research questions, of which this thesis could only address a few. To conclude this thesis, we now give an outlook on possible directions of future research that we identified during our work on the topic of resource-aware plan recognition in instrumented environments.

- **Investigation of Relevant Performance Factors**

  An interesting question for further research is, if (and how) different properties of the assumed knowledge models (plan library, sensor model, support model, cost-reward model) influence the performance of the proposed utility model and sensor selection strategies. While it seems obvious that the size (number of states, number of actions, etc.) of the plan library, sensor model, and support model directly influences the resulting runtime performance of the presented sensor selection strategies, other properties like the structure of the plan library might

influence the general effectiveness of sensor selection. Relevant factors here might be for instance the number/length of plans and plan alternatives, the number of interconnections between plans, or the general connectedness/sparseness of the plan library and support graph.

Another factor that might influence the performance of the proposed models and algorithms is the possible inaccuracy of the assumed knowledge models. In plan recognition one generally assumes, that the given models describe the agent's behavior completely and exactly. In practical applications this assumption usually cannot be justified. The question then is how factual deviations from the assumed knowledge models influence the performance of the proposed utility model and derived sensor selection strategies.

- **Submodularity of Utility Model**

  Submodularity is an interesting property of certain utility functions and represents the principle of *diminishing returns*. Diminishing returns appear whenever adding an item to a large set yields equal or less additional utility than adding the same item to a smaller subset. Intuitively, this corresponds to the case of overlapping sensors where each combination of sensors contributes at most as much as the sum of single sensors. Formally, we say that a function $u : 2^S \mapsto \mathbb{R}$ is submodular for a finite set S iff $\forall A \subset B \subset S$ and $\forall s \in S \setminus B$ holds $u(A \cup \{s\}) - u(A) \geq u(B \cup \{s\}) - u(B)$.

  Submodularity is an interesting property for utility functions that are used for the purpose of utility-based sensor selection, as one can proof that for such functions a greedy algorithm can approximately maximize u with a constant factor of $(1 - 1/e)$ (approximately 63%) [NWF78]. Hence utility-based sensor selection problems with submodular utility functions can be efficiently approximated with a known performance guarantee.

  The graph of utility values discovered in our evaluation of the utility model (see Figure 10.4) suggest that the proposed utility model is submodular for the used knowledge models, as the first deviation of the utility value graph for DT-BNB monotonically decreases as more resources (respectively more sensors) are used. Hence adding additional resources/sensors has a larger impact on the resulting utility for smaller amounts of available resources than for larger amounts of available resources. However, a formal proof is required in order to understand if (and for which basic knowledge models) the desired property of submodularity holds with respect to the proposed utility model.

- **Support of Dynamic Environments**

  Our approach to the problem of resource-aware plan recognition in instrumented environments assumes that all relevant knowledge is contained within one plan library, sensor model, support model, and cost-reward model each. It further assumes that these models are more or less static; otherwise a possibly costly recompilation of the utilized POMDP is required after each change to these models. In most of today's existing application scenarios this is not a big issue, as the change rate of components, applications, and agent habits often is limited. However, the original Ubiquitous Computing vision thinks of smart mobile artifacts, which spontaneously cooperate in order to realize novel kinds of applications. One consequence of this vision is, that instrumented environments become highly dynamic, and that applications form ad-hoc and possibly only exist for a few minutes or even seconds.

  In such environments, the assumption of monolithic and stable knowledge models is no longer justified. The plans that might be executed by users in dynamic environments depend on the current context and in particular the present artifacts, hence the plan library is highly volatile and usually unknown in advance. Plan recognition systems have to account for this factor by using techniques which allow for constructing the plan library on the fly, for instance by applying techniques that synthesize concrete plans from abstract goal descriptions and artifact-based task models.

  In order to apply approaches like the proposed utility model and derived sensor selection strategies in highly dynamic environments, we have to explore ways to dynamically assemble the required knowledge models from fragmentary smaller models, which might for instance be attached to objects, agent, and places. We further have to figure out how to calculated expected utility of information more efficiently, e.g. by iteratively refining POMDP policies on changes, or by using techniques similar to case-based reasoning in order to combine existing smaller utility models to new larger utility models.

- **Acquisition of Knowledge Models**

  A critical aspect for the successful application of the approaches that we proposed in this thesis is the availability of correct and complete knowledge models. In section 8.3 we already shared some thoughts on how these models could be acquired in real-world application scenarios. In order to realize the presented (or similar) ideas, one has to

investigate aspects related to knowledge representation and knowledge engineering, machine learning, user modeling, and automated knowledge driven model generation.

In a second step, the practical applicability of the identified theoretical methods has to be evaluated in different real-world scenarios. For this purpose one can use instrumented environments like the *Smart Kitchen* environment that we presented in section 9.1. Via sensors installed in such environments, raw observation data about the user's behavior can be collected, which can then be used to test and evaluate techniques for the learning of plan-recognition-related knowledge models.

Other approaches may extend and practically test systems and methods like Stahl's YAMAMOTO toolkit or his GOAL methodology (see subsection 8.3.1), which allow for a manual creation of required knowledge models through a domain expert.

- **Expressiveness of Underlying Theoretical Models**

  The complexity and structure of plans that are recognizable by a plan recognition system in general depends on the expressiveness of the theoretical models that are used to represent plan knowledge and to perform reasoning over this knowledge. Thereby, different theoretical models have different strengths and weaknesses. The models and theories we used in the context of this thesis for instance assume that the agent's plan selection and execution process is a discrete-time Markov chain, which allows for a convenient and efficient use of techniques like DBNs and POMDPs. On the downside, it makes the representation of plans with long-duration actions, loops with non-geometrically distributed numbers of iterations, recursive sub plans, and partially ordered plans cumbersome, as we discussed in section 8.4.

  Hence a worthwhile direction of future research is to investigate other theoretical frameworks and formalisms for the purpose of state-aware plan recognition and the associated problem of sensor selection. Such frameworks might for instance allow for the recognition of multiple plans that are executed in parallel (our framework does only allow for the recognition of single plans), support the recognition of cooperative multi-agent plan execution, or naturally represent continuous time plan execution processes and/or partially ordered and hierarchical plans.

  An important aspect to bear in mind if other theoretical frameworks are investigated is to consider their ability to serve as a foundation for a utility model of observation information. This is important in order

to apply the plan recognition approaches resulting from these models in conjunction with sensor selection techniques as demonstrated in this thesis. This might require applying novel approaches to compute extrinsic utility of observation information; including the use of other/additional knowledge models to represent support actions respectively costs and rewards, or might even require the application of other utility measures besides extrinsic utility.

- **Evaluation Methods for Plan Recognition Systems**

  An important corner stone of high quality scientific research is a reproducible and comparable evaluation of the obtained results. In fields like computer vision or speech recognition over time a set of general evaluation methods has evolved for this purpose, including standard corpora of test data, well defined evaluation criteria, and common ways to describe and set up reproducible experiments. Today, similar methods are virtually none-existent for research in the field of plan recognition (similar holds for the fields of event and activity recognition)[1].

  Although some of the evaluation methods that are applied to other types of pattern recognition problems can be applied to the field of plan recognition, a direct transfer is difficult or even impossible due to specific characteristics of the plan recognition domain. These include difficulties in the acquisition of meaningful and sufficiently large sets of observation data, the strong impact of the current context on the observed agent's behavior and the resulting observations (including the general problem of distinguishing between relevant and irrelevant context), the lack of a standard representation for observation, action, and plan information due to the broad range of possible levels of abstraction and constraints that may be used to describe plan libraries and sensor models (including temporal, spatial, and contextual constraints), and finally the inherently complex structure of autonomous agents' behavior, in particular in multi-agent domains with parallel plans.

  Hence, future research should also aim at developing and establishing standard evaluation methods which allow to judge and compare the performance of plan recognition approaches in practically relevant application settings in a repeatable manner . For the reasons summarized above, these methods have to be particularly designed to meet the special challenges that arise in the context of plan recognition applications. Possible general evaluation methods could for instance be based on a

[1] A notable exception is [BA05].

simulation approach similar to the approach that we presented in the evaluation part of this thesis.

The next step then is to promote the use of standardized evaluation methods. Here, one possibility could be to organize a plan recognition competition, which poses a challenge that hopefully stimulates the development of better and novel plan recognition systems while at the same time providing researchers with an incentive to practically apply standardized evaluation methods in their research.

Bibliography

[AA07] Marcelo G. Armentano and Analía Amandi. Plan recognition for interface agents - State of the art. *Artificial Intelligence Review*, 28(2):131–162, 2007.

[AFPB09] Hamdi Aloulou, Mohamed A. Feki, Clifton Phua, and Jit Biswas. Efficient incremental plan recognition method for cognitive assistance. In *Proceedings of the 7th International Conference on Smart Homes and Health Telematics (ICOST), LNCS 5597*, pages 225–228. Berlin, Heidelberg, Germany: Springer, 2009.

[AVH09] Javier Andréu, Jaime Viúdez, and Juan A. Holgado. An ambient assisted-living architecture based on wireless sensor networks. In *Proceedings of the 3rd Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI 2008)*, pages 239–248. Berlin, Heidelberg, Germany: Springer, 2009.

[AZN98] David W. Albrecht, Ingrid Zukerman, and Ann E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8(1–2):5–47, 1998.

[BA05] Nate Blaylock and James Allen. Generating artificial corpora for plan recognition. In *Proceedings of the 10th International Conference on User Modeling (UM), LNCS 3538*, pages 179–188. Berlin, Heidelberg, Germany: Springer, 2005.

[Bau94] Mathias Bauer. Integrating probabilistic reasoning into plan recognition. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, pages 620–624. Chichester, UK: Wiley, 1994.

[Bau99] Mathias Bauer. From interaction data to plan libraries: A clustering approach. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 962–967. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[BBG06] Bruno Bouchard, Abdenour Bouzouane, and Sylvain Giroux. A smart home agent for plan recognition. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 320–322. New York, NY, USA: ACM, 2006.

[BCF05] Kevin W. Bowyer, Kyong I. Chang, and Patrick J. Flynn. A survey of approaches and challenges in 3D and multi-modal 3D-2D face recognition. *Computer Vision and Image Understanding*, 101(1):10–15, 2005.

[BG95] Fahiem Bacchus and Adam Grove. Graphical models for preference and utility. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 3–10. San Francisco, CA, USA: Morgan Kaufman, 1995.

[BHK+06] Tatiana Bokareva, Wen Hu, Salil Kanhere, Branko Ristic, Travis Bessell, Mark Rutten, and Sanjay Jha. Wireless sensor networks for battlefield surveillance. In *Proceedings of The Land Warfare Conference (LWC)*, 2006.

[BJ04] Boris Brandherm and Anthony Jameson. An extension of the differential approach for Bayesian network inference to dynamic Bayesian networks. *International Journal of Intelligent Systems*, 19(8):727–748, 2004.

[BJH+07] B. Banks, G. Jackson, J. Helly, D. Chin, T.J. Smith, A. Schmidt, P. Brewer, R. Medd, D. Masters, A. Burger, and W.K. Krebs. Using behavior analysis algorithms to anticipate security threats before they impact mission critical operations. *Proceedings of the International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 307–312, 2007.

[BKG06] Fang Bian, David Kempe, and Ramesh Govindan. Utility-based sensor selection. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 11–18. New York, NY, USA: ACM, 2006.

[BKR+06] Michael Beigl, Albert Krohn, Till Riedel, Tobias Zimmer, Christian Decker, and Manabu Isomura. The μPart experience: Building a wireless sensor network. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 366–373. New York, NY, USA: ACM, 2006.

[BKSB08] Mathias Bauer, Alexander Kröner, Michael Schneider, and Nathalie Basselin. *Intelligent User Interfaces: Adaptation and Personalization Systems and Technologies*, chapter Building Digital Memories for Augmented Cognition and Situated Support. Hershey, PA, USA: IGI Global, 2008.

[BMG+09] Kai Breiner, Oliver Maschino, Daniel Görlich, Gerrit Meixner, and Detlef Zühlke. Run-time adaptation of a universal user interface for ambient intelligent production environments. In *Proceedings of the 13th International Conference on Human-Computer Interaction (HCI), LNCS 5613*, pages 663–672. Berlin, Heidelberg, Germany: Springer, 2009.

[Bou02] Craig Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, pages 239–246. Menlo Park, CA, USA: AAAI Press, 2002.

[BS05] Boris Brandherm and Tim Schwartz. Geo referenced dynamic Bayesian networks for user positioning on mobile systems. In *Proceedings of the 1st International Workshop on Location- and Context-Awareness (LoCA), LNCS 3479*, pages 223–234. Berlin, Heidelberg, Germany: Springer, 2005.

[BSS04] Andreas Butz, Michael Schneider, and Mira Spassova. SearchLight - A lightweight search function for pervasive environments. In *Proceedings of the 2nd International Conference on Pervasive Computing (Pervasive), LNCS 3001*, pages 351–356. Berlin, Heidelberg, Germany: Springer, 2004.

[Bui02] Hung H. Bui. Efficient approximate inference for online probabilistic plan recognition. In *Proceedings of the AAAI Fall Symposium on Intent Inference for Users, Teams and Adversaries*, pages 25–32. Menlo Park, CA, USA: AAAI Press, 2002.

[Bui03] Hung H. Bui. A general model for online probabilistic plan recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1309–1318. San Francisco, CA, USA: Morgan Kaufmann, 2003.

[Cac03] Dan G. Cacuci. *Sensitivity and Uncertainty Analysis: Theory.* London, UK: Chapman & Hall, 2003.

[Car01] Sandra Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.

[CCCT05] Wenjie Chen, Lifeng Chen, Zhanglong Chen, and Shiliang Tu. A realtime dynamic traffic control system based on wireless sensor network. In *Proceedings of the 35th International Conference on*

Parallel Processing Workshops (ICPPW), pages 258–264. Washington, DC, USA: IEEE Computer Society, 2005.

[CD93] James L. Crowley and Yves Demazeau. Principles and techniques for sensor data fusion. *Signal Processing*, 32(1-2):5–27, 1993.

[CE07] Sandra Carberry and Stephanie Elzer. Exploiting evidence analysis in plan recognition. In *Proceedings of the 11th International Conference on User Modeling (UM), LNCS 4511*, pages 7–16. Berlin, Heidelberg, Germany: Springer, 2007.

[CG93] Eugene Charniak and Robert P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[CKL94] Anthony R. Cassandra, Leslie P. Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, pages 1023–1028. Menlo Park, CA, USA: AAAI Press, 1994.

[CKP00] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, pages 363–369. Menlo Park, CA, USA: AAAI Press, 2000.

[CLZ97] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 54–61. San Francisco, CA, USA: Morgan Kaufmann, 1997.

[CR96] George Casella and Christian P. Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

[Cro09a] Crossbow Technology Inc., San Jose, CA, USA. *Imote2 Datasheet*, 2009.

[Cro09b] Crossbow Technology Inc., San Jose, CA, USA. *MICAz Datasheet*, 2009.

[DB02] Joachim Denzler and Christopher M. Brown. An information theoretic approach to optimal sensor data selection for state estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:145–157, 2002.

[Dem68] Arthur P. Dempster. A generalization of Bayesian inference. *Journal of the Royal Statistical Society, Series B*, 30:205–247, 1968.

[DLT02] Rami Debouk, Stéphane Lafortune, and Demosthenis Teneketzis. On an optimization problem in sensor selection. *Discrete Event Dynamic Systems*, 12(4):417–445, 2002.

[DP99] Oskar Dressler and Frank Puppe. Knowledge-based diagnosis – survey and future directions. In *Proceedings of the 5th Biannual German Conference on Knowledge Based Systems (XPS), LNAI 1590*, pages 24–46. Berlin, Heidelberg, Germany: Springer, 1999.

[FC03] Michael Fagan and Pádraig Cunningham. Case-based plan recognition in computer games. In *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR), LNAI 2689*, pages 161–170. Berlin, Heidelberg, Germany: Springer, 2003.

[Flo67] Robert W. Floyd. Nondeterministic algorithms. *Journal of the ACM (JACM)*, 14(4):636–644, 1967.

[Fre86] Simon French. *Decision Theory*. New York, NY, USA: Halsted, 1986.

[GCHM06] Vijay Gupta, Timothy H. Chung, Babak Hassibi, and Richard M. Murray. On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage. *Automatica*, 42(2):251–260, 2006.

[Gei04] Christopher W. Geib. Assessing the complexity of plan recognition. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, pages 507–512. Menlo Park, CA, USA: AAAI Press, 2004.

[Ger09a] German Research Center for Artificial Intelligence (DFKI). Bremen Ambient Assisted Living Lab (BAAL). Website: `http://www.baall.net/`, September 2009.

[Ger09b] German Research Center for Artificial Intelligence (DFKI). Innovative Retail Laboratory (IRL). Website: `http://www.dfki.de/irl/`, September 2009.

[GG01] Christopher W. Geib and Robert P. Goldman. Plan recognition in intrusion detection systems. In *Proceedings of the 2nd Information Survivability Conference and Exposition (DISCEX)*, pages 329–342, 2001.

[GPJ+08] Mario Gomez, Alun Preece, Matthew P. Johnson, Geeth de Mel, Wamberto Vasconcelos, Christopher Gibson, Amotz Bar-Noy, Konrad Borowiecki, Thomas La Porta, Diego Pizzocaro, Hosam Rowaihy, Gavin Pearson, and Tien Pham. An ontology-centric approach to sensor-mission assignment. In *Proceedings of the 16th International Conference on Knowledge Engineering: Practice and Patterns (EKAW), LNCS 5268*, pages 347–363. Berlin, Heidelberg, Germany: Springer, 2008.

[Gre03] Amy Greenwald. The 2002 trading agent competition: An overview of agent strategies. *AI Magazine*, 24(1):83–91, 2003.

[Hau00] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[Hec93] Matthias Hecking. *Eine logische Behandlung der verteilten und mehrstufigen Planerkennung.* Aachen, Germany: Shaker, 1993.

[Hec05] Dominik Heckmann. *Ubiquitous User Modeling*, volume 297 of *DISKI.* Berlin, Germany: Akademische Verlagsgesellschaft Aka, 2005.

[HGP99] Clinton Heinze, Simon Goss, and Adrian Pearce. Plan recognition in military simulation: Incorporating machine learning with intelligent agents. In *Proceedings of the IJCAI-99 Workshop on Team Behavior and Plan Recognition*, pages 53–64, 1999.

[HJSS06] Jon C. Helton, Jay D. Johnson, Cedric J. Salaberry, and Curt B. Storlie. Survey of sampling based methods for uncertainty and sensitivity analysis. *Reliability Engineering and System Safety*, 91:1175–1209, 2006.

[HKB+07] Thomas Haenselmann, Thomas King, Marcel Busse, Wolfgang Effelsberg, and Markus Fuchs. Scriptable sensor network based home-automation. In *Proceedings of the Embedded and Ubiquitous Computing Workshops (EUC), LNCS 4809*, pages 579–591. Berlin, Heidelberg, Germany: Springer, 2007.

[HM97] Geir E. Hovland and Brenan J. McCarragher. Dynamic sensor selection for robotic systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Vol. 1*, pages 272–277. Washington, DC, USA: IEEE Computer Society, 1997.

[HRS89] Gerd Herzog and Gudula Retz-Schmidt. Das System SOCCER: Simultane Interpretation und natürlichsprachliche Beschreibung zeitveränderlicher Szenen. In J. Pearl, editor, *Sport & Informatik*, pages 95–119. Schorndorf, Germany: Hofmann, 1989.

[HV99] Kwun Han and Manuela Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of the 9th International Symposium of Robotics Research (ISSR)*, pages 199–204. Berlin, Heidelberg, Germany: Springer, 1999.

[IB05] Volkan Isler and Ruzena Bajcsy. The sensor selection problem for bounded uncertainty sensing models. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 372–381. Washington, DC, USA: IEEE Computer Society, 2005.

[JKBS07] Anthony Jameson, Alexander Kröner, Nathalie Basselin, and Michael Schneider. Memory matching in support of interpersonal communication. In *CHI 2007 Workshop on Shared Encounters*, 2007.

[JKG03] Shengbing Jiang, Ratnesh Kumar, and Humberto E. Garcia. Optimal sensor selection for discrete-event systems with partial observation. *IEEE Transactions on Automatic Control*, 48(3):369–381, 2003.

[JLM04] Peter A. Jarvis, Teresa F. Lunt, and Karen L. Myers. Identifying terrorist activity with ai plan recognition technology. In *Proceedings of the 16th Conference on Innovative Applications of Artifical Intelligence (IAAI)*, pages 858–863. Menlo Park, CA, USA: AAAI Press, 2004.

[Jun08] Ralf Jung. Information transfer efficiency of peripheral audio cues. In *Proceedings of 4th International Conference on Intelligent Environments (IE)*. Stevenage, UK: IET, 2008.

[Jun09] Ralf Jung. *Ambiente Audionotifikation – Ein System zur kontextsensitiven Integration von nicht-intrusiven Notifikationssignalen*

in emotionsklassifizierten Soundscapes. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2009.

[Kah07] Gerrit Kahl. PEG und MaMiNa: Erkennung von Zeigegesten in einer instrumentierten Umgebung und auf Landmarken basierte Wegsuche zur Navigation. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2007.

[Kau91] Henry A. Kautz. A formal theory of plan recognition and its implementation. In J. F. Allen, H. A. Kautz, R. N. Pelavin, and J. D. Tennenberg, editors, *Reasoning About Plans*, pages 69–126. San Fransisco, CA, USA: Morgan Kaufmann, 1991.

[KB04] Hirokazu Kato and Mark Billinghurst. Developing AR applications with ARToolKit. In *Proceedings of the 3rd International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 305–305. Washington, DC, USA: IEEE Computer Society, 2004.

[KBSM07] Alexander Kröner, Nathalie Basselin, Michael Schneider, and Junichiro Mori. Selecting users for sharing augmented personal memories. In *Proceedings of the 30th Annual German Conference on AI (KI), LNAI 4667*, pages 477–480. Berlin, Heidelberg, Germany: Springer, 2007.

[KC01] Boris Kerkez and Michael T. Cox. Case-based plan recognition using state indices. In *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR), LNAI 2080*, pages 291–305. Berlin, Heidelberg, Germany: Springer, 2001.

[KC03] Boris Kerkez and Michael T. Cox. Incremental case-based plan recognition with local predictions. *International Journal on Artificial Intelligence Tools: Architectures, languages, algorithms*, 12(4):413–464, 2003.

[KHS06] Alexander Kröner, Dominik Heckmann, and Michael Schneider. Exploiting the link between personal, augmented memories and ubiquitous user modeling. In *Proceedings of the ECAI 2006 Workshop on Ubiquitous User Modelling (UbiqUM)*, pages 25–26, 2006.

[KHW06] Alexander Kröner, Dominik Heckmann, and Wolfgang Wahlster. SPECTER: Building, exploiting, and sharing augmented memories. In *Proceedings of the Workshop on Knowledge Sharing for Everyday Life (KSEL)*, pages 9–16. Kyoto, Japan: ATR, 2006.

[Kja92] Uffe Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the 8th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 121–129. San Francisco, CA, USA: Morgan Kaufmann, 1992.

[KJSB08] Alexander Kröner, Anthony Jameson, Michael Schneider, and Nathalie Basselin. Augmenting cognition with a digital episodic memory. *Künstliche Intelligenz (KI)*, 22(2):51–57, 2008.

[KP98] Michael Kalandros and Lucy Y. Pao. Controlling target estimate covariance in centralized multisensorsystems. In *Proceedings of the American Control Conference (ACC), Vol. 5*, pages 2749–2753. Washington, DC, USA: IEEE Computer Society, 1998.

[KPP05] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Berlin, Heidelberg, Germany: Springer, 2005.

[KSM+07] Matthias Kranz, Albrecht Schmidt, Alexis Maldonado, Radu Bogdan Rusu, Michael Beetz, Benedikt Hörnler, and Gerhard Rigoll. Context-aware kitchen utilities. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI)*, pages 213–214. New York, NY, USA: ACM, 2007.

[KSM09] Alexander Kröner, Michael Schneider, and Junichiro Mori. A framework for ubiquitous content sharing. *IEEE Pervasive Computing*, 8(4):58–65, 2009.

[Leo78] Aleksei N. Leontiev. *Activity, Consciousness, and Personality*. Upper Saddle River, NJ, USA: Prentice Hall, 1978. Original work published in Russian in 1975.

[LFK05] Lin Liao, Dieter Fox, and Henry A. Kautz. Location-based activity recognition using relational markov networks. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 773–778. Denver, CO, USA: Professional Book Center, 2005.

[LKO08] Jaeyong Lee, Bonjung Koo, and Kyungwhan Oh. State space optimization using plan recognition and reinforcement learning on RTS game. In *Proceedings of the 7th International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED)*, pages 165–169. Stevens Point, WI, USA: WSEAS Press, 2008.

[LMP01] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 282–289. San Francisco, CA, USA: Morgan Kaufmann, 2001.

[LW66] Eugene L. Lawler and David E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14:699–719, 1966.

[LWG05] Olaf Landsiedel, Klaus Wehrle, and Stefan Götz. Accurate prediction of power consumption in sensor networks. In *Proceedings of The 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS)*. Washington, DC, USA: IEEE Computer Society, 2005.

[MBKJ08] Junichiro Mori, Nathalie Basselin, Alexander Kröner, and Anthony Jameson. Find me if you can: Designing Interfaces for People Search . In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, pages 377–380. New York, NY, USA: ACM, 2008.

[MBS+07] Junichiro Mori, Nathalie Basselin, Michael Schneider, Alexander Kröner, Anthony Jameson, and Wolfgang Wahlster. SharedLife: Sharing of augmented personal memories in ubiquitous environments. In *Proceedings of the 21st Annual Conference of the Japanese Society for Artificial Intelligence (JSAI), LNCS 4914*. Berlin, Heidelberg, Germany: Springer, 2007.

[MC99] Eric Marchand and Fran Cois Chaumette. An autonomous active vision system for complete and accurate 3D scene reconstruction. *Journal of Computer Vision*, 32:171–194, 1999.

[MCP+02] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97. New York, NY, USA: ACM, 2002.

[Mil56] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

[MLB+02] Lyudmila Mihaylova, Tine Lefebvre, Herman Bruyninckx, Klaas Gadeyne, and Joris De Schutter. Active sensing for robotics –

a survey. In *Proceedings of the 5th International Conference on Numerical Methods and Applications (NMA), LNCS 2542*, pages 316–324. Berlin, Heidelberg, Germany: Springer, 2002.

[MM04] Frank Manola and Eric Miller, editors. *RDF Primer*. W3C Recommendation. World Wide Web Consortium (W3C), February 2004.

[MM05] Carlos H. Morimoto and Marcio R. M. Mimica. Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98(1):4–24, 2005.

[Mos09] Clemens Moser. *Power Management in Energy Harvesting Embedded Systems*. Shaker, 2009.

[MPF+08] Filipe Martins, Joana Paulo Pardal, Luís Franqueira, Pedro Arez, and Nuno J. Mamede. Starting to cook a tutoring dialogue system. In *Proceedings of the 2nd IEEE/ACL Workshop on Spoken Language Technology (SLT)*, pages 145–148. Washington, DC, USA: IEEE Computer Society, 2008.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge, UK: Cambridge University Press, 1995.

[MS99] Piero La Mura and Yoav Shoham. Expected utility networks. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 366–373. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[MSD08] Jörg Baus Michael Schmitz and Robert Dörr. The Digital Sommelier: Interacting with intelligent products. In *Proceedings of the 1st International Conference on the Internet of Things (IOT), LNCS 4952*, pages 247–262. Berlin, Heidelberg, Germany: Springer, 2008.

[MV03] Frank Mulder and Frans Voorbraak. A formal description of tactical plan recognition. *Information Fusion*, 4(1):47–61, 2003.

[Nag04] Hans-Hellmut Nagel. Steps toward a cognitive vision system. *AI Magazine*, 25(2):31–50, 2004.

[NGK+05] Alassane Ndiaye, Patrick Gebhard, Michael Kipp, Martin Klesen, Michael Schneider, and Wolfgang Wahlster. Ambient intelligence in edutainment: Tangible interaction with life-like exhibit guides. In *Proceedings of 1st International Conference*

on Intelligent Technologies for Interactive Entertainment (INTETAIN), LNAI 3814, pages 104–113. Berlin, Heidelberg, Germany: Springer, 2005.

[NWF78] George Nemhauser, Laurence Wolsey, and Marshall Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

[Oak04] Jeremy E. Oakley. Probabilistic sensitivity analysis of complex models: A Bayesian approach. *Journal of the Royal Statistical Society, Series B*, 66:751–769, 2004.

[OH04] Nuria Oliver and Eric Horvitz. S-seer: Selective perception in a multimodal office activity recognition system. In *Proceedings of the 1st International Workshop on Machine Learning for Multimodal Interaction (MLMI), LNCS 3361*, pages 122–135. Berlin, Heidelberg, Germany: Springer, 2004.

[Osh94] Yaakov Oshman. Optimal sensor selection strategy for discrete-time state estimators. *IEEE Transactions on Aerospace and Electronic Systems*, 30(2):307–314, 1994.

[OXMH09] Patrick Olivier, Guangyou Xu, Andrew Monk, and Jesse Hoey. Ambient kitchen: Designing situated services using a high fidelity prototyping environment. In *Proceedings of the 2nd International Conference on Pervasive Technologies Related to Assistive Environments (PETRA)*, pages 1–7. New York, NY, USA: ACM, 2009.

[PAG06] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. In *Proceedings of the 5th International Semantic Web Conference (ISWC), LNCS 4273*, pages 30–43. Berlin, Heidelberg, Germany: Springer, 2006.

[PBK+06] Carolin Plate, Nathalie Basselin, Alexander Kröner, Michael Schneider, Stephan Baldes, Vania Dimitrova, and Anthony Jameson. Recomindation: New functions for augmented memories. In *Proceedings of the 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH), LNCS 4018*. Berlin, Heidelberg, Germany: Springer, 2006.

[Pea85] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society (CogSci)*, pages 329–334.

Los Angeles, CA, USA: UCLA Computer Science Department, 1985.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann, 1988.

[PGT03] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025–1032. San Francisco, CA, USA: Morgan Kaufmann, 2003.

[PH93] K. L. Poh and E. J. Horvitz. Reasoning about the value of decision model refinement: Methods and application. In *Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 174–182. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[PKMT99] Avi Pfeffer, Daphne Koller, Brian Milch, and Ken T. Takusagawa. SPOOK: A System for probabilistic object-oriented knowledge representation. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 541–550. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[PW00] David V. Pynadath and Michael P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 507–514. San Francisco, CA, USA: Morgan Kaufmann, 2000.

[Rab63] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

[Rab89] Herschel Rabitz. Systems analysis at the molecular scale. *Science*, 13(4927):221–226, 1989.

[RBBG07] Patrice Roy, Bruno Bouchard, Abdenour Bouzouane, and Sylvain Giroux. A hybrid plan recognition model for Alzheimer's patients: Interleaved-erroneous dilemma. In *Proceedings of the 6th International Conference on Intelligent Agent Technology (IAT)*, pages 131–137. Washington, DC, USA: IEEE Computer Society, 2007.

[RBP+06] Ricardo Ribeiro, Fernando Batista, Joana Paulo Pardal, Nuno J. Mamede, and H. Sofia Pinto. Cooking an ontology. In *Proceedings of the 12th International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA), LNCS 4183*, pages 213–221. Berlin, Heidelberg, Germany: Springer, 2006.

[REJ+07] Hosam Rowaihy, Sharanya Eswaran, Matthew Johnson, Dinesh Verma, Amotz Bar-Noy, Theodore Brown, and Thomas La Porta. A survey of sensor selection schemes in wireless sensor networks. In *Proceedings of the Defense and Security Symposium on Unattended Ground, Sea, and Air Sensor Technologies and Applications IX (DSS), SPIE 6562*, page 65621A. Bellingham, WA, USA: SPIE, 2007.

[RGB08] Radu Bogdan Rusu, Brian P. Gerkey, and Michael Beetz. Robots in the kitchen: Exploiting ubiquitous sensing and actuation. *Robotics and Autonomous Systems*, 56(10):844–856, 2008.

[RM94] Anand S. Rao and Graeme Murray. Multi-agent mental-state recognition and its application to air-combat modeling. In *Proceedings of the 13th International Distributed Artificial Intelligence Workshop (DAI Workshop)*, pages 283–304. Menlo Park, CA, USA: AAAI Press, 1994.

[RM04] Kay Römer and Friedemann Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, 2004.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter Probabilistic Reasoning. Upper Saddle River, NJ, USA: Prentice Hall, 3rd edition, 2009.

[Ros83] Sheldon M. Ross. *An Introduction to Stochastic Dynamic Programming*. New York, NY, USA: Academic Press, 1983.

[RS92] Gudula Retz-Schmidt. *Die Interpretation des Verhaltens mehrerer Akteure in Szenenfolgen*. Berlin, Heidelberg, Germany: Springer, 1992.

[SB05] Michael Schneider and Andreas Butz. Wipe it! A direct manipulation technique for ubiquitous information items. In *Proceedings of the International Workshop on Intelligent Environments (IE)*, pages 168–172. Stevenage, UK: IET, 2005.

[SBK05a] Michael Schneider, Mathias Bauer, and Alexander Kröner. Building a personal memory for situated user support. In *Proceedings of the Pervasive 2005 Workshop on Exploiting Context Histories in Smart Environments (ECHISE), CSRP 557*, pages 43–48. Brighton, UK: University of Sussex, 2005.

[SBK05b] Michael Schneider, Andreas Butz, and Antonio Krüger. Flash and peep: A robust method for finding and tracking displays. In *Proceedings of the International Workshop on Intelligent Environments (IE)*, pages 192–198. Stevenage, UK: IET, 2005.

[Sch96] Karl Schäfer. *Unscharfe zeitlogische Modellierung von Situationen und Handlungen in Bildfolgenauswertung und Robotik*, volume 135 of *DISKI*. St. Augustin, Germany: Infix, 1996.

[Sch03] Michael Schneider. OPRES: Ein System zur Planerkennung in dynamischen instrumentierten Umgebungen. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany, 2003.

[Sch04] Michael Schneider. Towards a transparent proactive user interface for a shopping assistant. In *Proceedings of the IUI 2004 Workshop on Multi-User and Ubiquitous User Interfaces (MU3I)*, pages 10–15. Saarbrücken, Germany: Universität des Saarlandes, 2004.

[Sch06] Michael Schneider. RDF:Stores - A lightweight approach on managing shared knowledge. In *Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC), LNCS 4159*, pages 168–172. Berlin, Heidelberg, Germany: Springer, 2006.

[Sch07a] Michael Schneider. The Semantic Cookbook: Sharing cooking experiences in the smart kitchen. In *Proceedings of the 3rd International Conference on Intelligent Environments (IE)*, pages 416–423. Stevenage, UK: IET, 2007.

[Sch07b] Michael Schneider. Towards a general object memory. In Anne Bajart, Henk Muller, and Thomas Strang, editors, *UbiComp 2007 Workshops Proceedings*, pages 307–312, 2007.

[SH06] Christoph Stahl and Jens Haupert. Taking location modelling to new levels: A map modelling toolkit for intelligent environments. In *Proceedings of the International Workshop on Location- and*

Context-Awareness (LoCA), LNCS 3987, pages 74–85. Berlin, Heidelberg, Germany: Springer, 2006.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[Sha76] Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton, NJ, USA: Princeton University Press, 1976.

[SHDC06] Randall B. Smith, Bernard Horan, John Daniels, and Dave Cleal. Programming the world with Sun SPOTs. In *Companion to the 21st Symposium on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 706–707. New York, NY, USA: ACM, 2006.

[SHKF09] Peter Stephan, Ines Heck, Peter Kraus, and Georg Frey. Evaluation of indoor positioning technologies under industrial application conditions in the SmartFactoryKL based on EN ISO 9283. In *Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM)*, pages 874–879. Moscow, Russia: ICS/RAS, 2009.

[Sho97] Yoav Shoham. A symmetric view of utilities and probabilities. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1324–1329. San Francisco, CA, USA: Morgan Kaufman, 1997.

[SKS07] Michael Schmitz, Antonio Krüger, and Sarah Schmidt. Modelling personality in voices of talking products through prosodic parameters. In *Proceedings of the 10th International Conference on Intelligent User Interfaces (IUI)*, pages 313–316. New York, NY, USA: ACM, 2007.

[SKSM07] Nico Schlitter, Florian Kähne, Stiefen T. Schilz, and Holger Mattke. *Innovative Logistics Management*, volume 4 of *Operations and Technology Management*, chapter Potential and Problems of RFID-Based Cooperation in a Supply Chain, pages 147–164. Berlin, Germany: Erich Schmidt Verlag, 2007.

[SKW06] Michael Schneider, Alexander Kröner, and Rainer Wasinger. Augmenting interaction in intelligent environments through Open Personal Memories. In *Proceedings of the 2nd International Conference on Intelligent Environments (IE)*, pages 407–416. Stevenage, UK: IET, 2006.

[SLL+08] Hongjoong Sin, Sungju Lee, Jangsu Lee, Seunghwan Yoo, Sanghyuc Lee, Jaesik Lee, and Sungchun Kim. Self-organized cluster based multi-hop routing for wireless sensor networks. In *Proceedings of the 11th Asia-Pacific Symposium on Network Operations and Management (APNOMS), LNCS 5297*, pages 499–502. Berlin, Heidelberg, Germany: Springer, 2008.

[Sma09] Technologie-Initiative SmartFactoryKL. Project description. Website: `http://www.smartfactory-kl.de/`, September 2009.

[Smi09] Trey Smith. ZMDP Software for POMDP and MDP Planning. `http://www.cs.cmu.edu/~trey/zmdp/`, September 2009. Version 1.1.7.

[SPSKP06] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.

[SS05] Trey Smith and Reid G. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 542–549. Arlington, VA, USA: AUAI Press, 2005.

[Sta09] Christoph Stahl. *Spatial Modeling of Activity and User Assistance in Instrumented Environments*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2009.

[STCR04] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, and Marco Ratto. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Hoboken, NJ, USA: Wiley, 2004.

[Thi94] Markus A. Thies. *Planbasierte Hilfeverfahren für direktmanipulative Systeme: Erkennung, Vervollständigung und Visualisierung von Interaktionsplänen*, volume 67 of *DISKI*. Sankt Augustin, Germany: Infix, 1994.

[TIL04] Emmanuel Munguia Tapia, Stephen S. Intille, and Kent Larson. Activity recognition in the home using simple and ubiquitous sensors. In *Proceedings of the 2nd International Conference on Pervasive Computing (Pervasive), LNCS 3001*, pages 158–175. Berlin, Heidelberg, Germany: Springer, 2004.

[Wae96] Annika Waern. *Recognising Human Plans: Issues for Plan Recognition in Human-Computer Interaction*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1996.

[Wah03a] Wolfgang Wahlster. SmartKom: Symmetric multimodality in an adaptive and reusable dialogue shell. In *Proceedings of the Human Computer Interaction Status Conference*, pages 47–62. Berlin, Germany: DLR, 2003.

[Wah03b] Wolfgang Wahlster. Towards symmetric multimodality: Fusion and fission of speech, gesture, and facial expression. In *Proceedings of the 26th German Conference on Artificial Intelligence (KI), LNAI 2821*, pages 1–18. Berlin, Heidelberg, Germany: Springer, 2003.

[WBG92] Michael P. Wellman, John S. Breese, and Robert P. Goldman. From knowledge bases to decision models. *Knowledge Engineering Review*, 7(1):35–53, 1992.

[Wei91] Mark Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.

[Wel82] William J. Welch. Branch-and-bound search for experimental designs based on d-optimality and other criteria. *Technometrics*, 24(1):41–48, 1982.

[Whi91] Chelsea C. White. A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research*, 32(1):215–230, 1991.

[WHK88] Wolfgang Wahlster, Matthias Hecking, and Christel Kemke. SC: Ein intelligentes Hilfesystem für SINIX. In *Innovative Informations-Infrastrukturen*, pages 81–100. Berlin, Heidelberg, Germany: Springer, 1988.

[Wil78] Robert Wilensky. Why john married mary: Understanding stories involving recurring goals. *Cognitive Science*, 2:235–266, 1978.

[Wil83] Robert Wilensky. *Planning and Understanding*. Boston, MA, USA: Addison-Wesley, 1983.

[WKH06] Wolfgang Wahlster, Alexander Kröner, and Dominik Heckmann. SharedLife: Towards selective sharing of augmented personal

memories. In *Reasoning, Action and Interaction in AI Theories and Systems, LNAI 4155*, pages 327–342. Berlin, Heidelberg, Germany: Springer, 2006.

[WKSB08] Wolfgang Wahlster, Alexander Kröner, Michael Schneider, and Jörg Baus. Sharing memories of smart products and their consumers in instrumented environments. *it – Information Technology*, 50(1):45–50, 2008.

[WKY+09] Zhaowen Wang, Ercan E. Kuruoglu, Xiaokang Yang, Yi Xu, and Songyu Yu. Event recognition with time varying Hidden Markov Model. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1761–1764. Washington, DC, USA: IEEE Computer Society, 2009.

[WS04] Rainer Wasinger and Michael Schneider. Multimodal interactions with an instrumented shelf. In *Proceedings of the UbiComp 2004 Workshop on Artificial Intelligence in Mobile Systems (AIMS)*, pages 36–43. Saarbrücken, Gemany: Universität des Saarlandes, 2004.

[WT97] Wolfgang Wahlster and Werner Tack. SFB 378: Ressourcenadaptive kognitive Prozesse. In *Informatik '97: Informatik als Innovationsmotor*, pages 51–57. Berlin, Heidelberg, Germany: Springer, 1997.

[WYPE04] Hanbiao Wang, Kung Yao, Greg Pottie, and Deborah Estrin. Entropy-based sensor selection heuristic for target localization. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 36–45. New York, NY, USA: ACM, 2004.

[YSK93] Leehter Yao, William A. Sethares, and Daniel C. Kammer. Sensor placement for on-orbit modal identification via a genetic algorithm. *American Institute of Aeronautics and Astronautics Journal*, 31(10):1922–1928, 1993.

[Zig08] ZigBee Alliance Inc, San Ramon, CA, USA. *ZigBee Specification*, 2008.