Linköping Studies in Science and Technology

Thesis No. 1363

Dynamic Abstraction for Interleaved Task Planning and Execution

by

Per Nyblom

Submitted to Linköping Institute of Technology at Linköping University in partial fulfilment of the requirements for degree of Licentiate of Engineering

Department of Computer and Information Science
Linköping universitet
SE-581 83 Linköping, Sweden

Linköping 2008

Dynamic Abstraction for Interleaved Task Planning and Execution

by

Per Nyblom

April 2008
ISBN 978-91-7393-905-8
Linköping Studies in Science and Technology
Thesis No. 1363
ISSN 0280–7971
LiU–Tek–Lic–2008:21

ABSTRACT

It is often beneficial for an autonomous agent that operates in a complex environment to make use of different types of mathematical models to keep track of unobservable parts of the world or to perform prediction, planning and other types of reasoning. Since a model is always a simplification of something else, there always exists a tradeoff between the model's accuracy and feasibility when it is used within a certain application due to the limited available computational resources. Currently, this tradeoff is to a large extent balanced by humans for model construction in general and for autonomous agents in particular. This thesis investigates different solutions where such agents are more responsible for balancing the tradeoff for models themselves in the context of interleaved task planning and plan execution. The necessary components for an autonomous agent that performs its abstractions and constructs planning models dynamically during task planning and execution are investigated and a method called DARE is developed that is a template for handling the possible situations that can occur such as the rise of unsuitable abstractions and need for dynamic construction of abstraction levels. Implementations of DARE are presented in two case studies where both a fully and partially observable stochastic domain are used, motivated by research with Unmanned Aircraft Systems. The case studies also demonstrate possible ways to perform dynamic abstraction and problem model construction in practice.

This work has been supported by the Swedish Aeronautics Research Council (NFFP4-S4203), the Swedish National Graduate School in Computer Science (CUGS), the Swedish Research Council (50405001) and the Wallenberg Foundation (WITAS Project).

Department of Computer and Information Science
Linköping universitet
SE-581 83 Linköping, Sweden

Acknowledgements

I would like to thank my advisor Patrick Doherty who has given me more or less free hands to investigate this fascinating field of Artificial Intelligence. It has truly been some of the most interesting years of my life and I apologize for always picking subjects that you are less familiar with.

During my time at the Artificial Intelligence and Integrated Computer Systems division (AIICS), I have received valuable input from many people. Special thanks to Martin Magnusson, Fredrik Heintz, Per-Magnus Olsson, David Landén, Piotr Rudol and Gianpaolo Conte for commenting drafts of this thesis and related papers at various (perhaps dynamically generated) levels of abstraction.

Thanks to Martin Magnusson for providing the fire for many interesting and sometimes endless discussions which really make me grow as a person. Also thanks to Patrik Haslum for your endless wisdom and for supporting me during my early development. Thanks to Fredrik Heintz for your sense of detail and perfection and Jonas Kvarnström for your incredible problem solving capabilities (and will to share them). Thanks to Tommy Person and Björn Wingman for your help with all the implementation issues and your insights into the UAS Tech system.

Finally, I thank my parents Kurt and Gunilla, my girlfriend Anna and my daughter Anneli for love and support.

Contents

Chapter 1

Introduction

It is difficult to overestimate the importance of mathematical models in our modern society because of their common use in e.g. natural sciences and engineering diciplines for many different purposes. Many types of models exists today that can be used for a variety of tasks such as predicting weather, simulating vehicle dynamics, monitoring nuclear power reactors and verifying computer programs.

Models have also been used within the area of Artificial Intelligence (AI) to develop autonomous agents. It is widely considered that such agents should have models of their environments (and themselves) to make it possible to operate more successfully. The models can for example be used to keep track of the unobservable parts of the world, perform prediction [34], task planning [31] and other types of reasoning [10].

1.1 Models and Tradeoffs

One common trait for mathematical models used in practical applications is that it is not always beneficial (or even possible) to model every aspect of a system of study down to the smallest detail to get as accurate as possible. The problem is that there is always a *tradeoff between accuracy and feasibility* of a model that should be used for a certain application on a given architecture. There might be a demand for timely response of a system that prohibits long deliberation time which in turn can make a highly detailed, but computationally demanding model inappropriate for use in that particular domain. Although the computational resources that can be made available for different applications have been increasing exponentially since the dawn of electronic computers, there will always be a limit when a particular system is being developed and deployed. This means that one will always have to trade a model's accuracy for feasibility to get a reasonable performance in *any future system*, which is a fact that is often mentioned in the literature about practical mathematical modelling [44] [24].

1.2 Task Environments and Models

When a model is to be constructed for an autonomous agent, it is important to consider the *task environment* [63] in which the agent will operate. The complexity of the task environment can give significant hints about the different types of models that can be used for whatever the purpose of the model is.

A task environment, which can be either real or simulated, specifies:

- what the agent can do to the environment with its *actuators*,
- what information it can receive from its *sensors*,
- how the *environment* works and what it contains, and
- what is considered "good or bad" with the help of a *performance measure*

A task environment for an autonomous ground robot can e.g. specify that the actuators consist of a propulsion system and possibly a manipulator arm. Such agents are also typically equipped with sensors such as laser range scanners, cameras and sometimes collision sensors. The environment may consist of tables, chairs, walls, stairs etc., and its performance measure may be defined in terms of power consumption and the time to complete an assigned task (such as delivering a package).

A model that an agent uses should be closely connected to the task environment that the agent operates within. For example, if a model is going to be used for predicting the state of an autonomous agent's task environment depending on what actions it performs, it better include specifications of how the actuators, sensors and the surrondings work in order to be useful. Such a *task environment model* can not be too detailed due to the tradeoff between accuracy and feasibility.

A task environment or a model thereof can be classified according to some commonly used dimensions [63] which to a large extent determine how difficult it is to handle.

- **Fully Observable** or **Partially Observable**: If the agent's sensors can give access to all the relevant information in the environment it is called a *fully observable* task environment; otherwise the task environment is called *partially observable*.
- **Deterministic** or **Stochastic**: If the next state is completely determined by the current state and the action executed by the agent the task environment is called *deterministic*. If there are several possible outcomes of an action it is called a *stochastic* environment. The term *non-deterministic* is often used when outcomes do not have probabilities associated with them.

- **Episodic** or **Sequential**: In an *episodic* environment, the agent's current decision does not influence the performance of any future episode. All environments considered in this thesis will be *sequential* which means that the agent's current decision might influence the performance of the agent in future states.

- **Static** or **Dynamic**: A task environment which may change while the agent deliberates is called a *dynamic* environment; otherwise it is called *static*.

- **Discrete** or **Continuous**: A *continuous* environment contains elements that are more accurately described with continuous models involving real values instead of an enumerable set of values. Task environments that do not have any continuous elements are called *discrete*.

- **Single Agent** or **Multiagent**: A task environment where other external agents, besides the main agent itself, try to reach goals or maximize their utilities are called *multiagent*. If the external agents are better described without decision capabilities, or if no external agents exist, the environment can be considered *single agent*.

In this thesis, these dimensions are used to classify the *intrinsic* properties of a task environment. They are not assumptions that e.g. a designer of an agent can make. On the other hand, a designer can make assumptions that are reflected in the agent's task environment models that it is supposed to use. Constructed models that represent parts of a task environment must often be a simplification of the real thing and the different dimensions are then used to classify the model construction assumptions that are not already a property of the task environment. This will be discussed more in Section 1.4.

It is assumed that task environment models can be *simulated*. This means that different actions can be tested with the model which may result in one or several possible outcomes depending on whether the model is deterministic or not. Stochastic models can be simulated by pseudo random number generators.

A *task environment class* or *environment class* is a set of task environments with similar properties. An agent is often designed to operate in instances of a particular task environment class where e.g. the environment can contain a different number of objects and agents but most of the other properties or assumptions stay the same. In this thesis, the task environment instances in a particular environment class are assumed to have the same classification according to the previously mentioned dimensions and that the actuators and sensors are similarly modelled. Within a particular environment class, the types of the objects in the environment also stay the same but the number and initial conditions may vary in task environment instances associated with the class.

The next topic of this introduction will describe an example of a complex task environment class that has motivated much of the work with this thesis.

1.3 The UAS Tech System

This thesis is very much inspired by the UAS Tech's Unmanned Aircraft System (UAS), which currently consist of two autonomous Yamaha RMAX helicopters equipped with sophisticated software and control systems that have been developed in the past decade [17]. Many examples in this thesis and the case studies have this platform in mind due to the huge variety of tasks that can potentially be performed with such a system.

A task environment in this environment class (see Section 1.2) can consist of a varity of elements. A training area for rescue workers in Revinge (southern Sweden) is often used as a test flight area and that area contains (or can be modified to contain) buildings, different types of obstacles, roads, vehicles, landing spots, safety operators, ground stations where human operators can monitor the UASs, injured humans (simulated by Phd students) and in the future also fires, smoke sources and boxes that can be transported.

Depending on the task that is supposed to be performed by the system, the performance measure (see Section 1.2) is different. If the task is to take a set of pictures of a selected number of building structures, the performance measure could include the time it takes to perform the mission, the quality of the pictures and whether all the requested building structures were photographed. In the case of another standard mission where a UAS is looking for victims in a catastrophe area (or the like) the performance measure could include the number of injured people detected, number of false positives and the time taken to perform the mission.

The actuators of a UAS, for the purposes of this thesis, are considered to be the signals that control the helicopter's rotors, camera (IR and color) pan/tilt unit, wireless network and General Packet Radio Service (GPRS) communication. The communication through the wireless network is performed with the help of the Common Object Request Broker Architecture (CORBA) [59]. An actuator planned to be used in the future is an electromagnet attached via a winch system to the UAS which can be used to transport objects such as medical supplies.

On a UAS Tech unmanned helicopter, numerous sensors are mounted including a Global Positioning System (GPS), Inertial Measurement Unit (IMU) and altimeter for pose estimation, color and infrared cameras. The wireless network and the GPRS connection are also considered sensors.

One of the UAS Tech's unmanned helicopters is pictured in figure 1.1. Its computational capabilities are distributed among three onboard computers where each of them is used for image processing [32], flight and camera controller [11] and deliberative functionality such as path planning [61], geographic information system and the deliberative/reactive execution system

[18], respectively.

Figure 1.1: One of the UAS Tech's unmanned helicopters.

In order to perform complex tasks in complicated task environments (such as with the UAS Tech system), it is necessary to structure the execution of tasks and the representation of the environment in a suitable way with the help of different types of models. Since the UAS Tech system is situated in the real world, the task environments in this environment class are the most difficult ones according to the dimensions listed in Section 1.2. On the other hand, the models that the system uses of the task environments are always a simplification. Assumptions like full observability, determinism and single agent are common, quite accurate and useful in certain types of missions.

1.4 Abstractions

In this thesis, the following definition of an abstraction is used:

Definition 1.1
*An **abstraction** is a simplification of the physical world or a simplification of a model.*

An abstraction is a process that removes details and should expose the most essential features of the entity that it is applied to. This thesis is primarily concerned with abstractions of task environments which are more agent-centered and include definitions of performance measures.

Definition 1.2
*A **task environment abstraction** is a simplification of a task environment or a simplification of a task environment model.*

Task environment abstractions are the only abstractions that will be discussed in the rest of this thesis. Whenever an abstraction is mentioned, it is meant to refer to a task environment abstraction. Figure 1.2 illustrates how abstractions can be used to construct models from task environments or models thereof.

In order to reason about complex task environments, abstractions are often used to construct simplified models with. This is especially the case when the task environment is part of the physical world where no exact model exists. But even task environments that are extremly open-ended can be reasoned about by performing abstractions that are reasonably valid under many circumstances.

The UAS Tech system's different task environments are very complex and require abstractions. For example, the roadmap-based path planning module [61] uses a polygon representation of the environment to construct collision-free path segments that a UAS can fly. The polygons are just simplifications of the environment but the resulting paths are still very reliable. Another example where abstractions are used is in the control system whose design assumes that the helicopter system is linear, which is never the case for robotic systems, but it is still possible to control the helicopter reliably with standard techniques from control theory.

Abstractions for higher level reasoning about task environments can also be performed which could result in facts like "landed", "at position p", "object o is visible from p". These facts can then be used to support task planning and execution monitoring. If a UAS e.g. performs vehicle tracking, event descriptions such as "vehicle v turns left at intersection i" might be useful to summarize a situation or send high level information to other agents. In these cases, the abstraction performs a discretization of parts of the continuous task environment and summarizes the infinitely many states of the environment into a countable number of discrete ones.

It is also possible to perform abstractions on models to construct even more simplified models (See the upper part of Figure 1.2) that can e.g. be used for computing heuristic functions to guide problem solvers. The simplified problem model can often be solved much faster (depending on the abstraction) than the original model, and this technique is frequently used to solve classical planning models (See Section 1.5) and integer linear programming [65] (ILP) problems.

An abstraction that transforms a task environment into another, simplified task environment model can be analysed with the same type of dimensions (fully/partially observable etc) as real task environments. The resulting model type might be incapable of expressing stochastic and/or partially observable phenomena when in fact the real task environment has these properties. These dimensions are only a rough categorization of abstractions, but they they tend to be very important for deciding the type of the resulting model.

The thesis focuses on abstractions and the resulting models used for *task planning* where there exists a rich set of model types that have different capabilities of expressing properties of task environments.

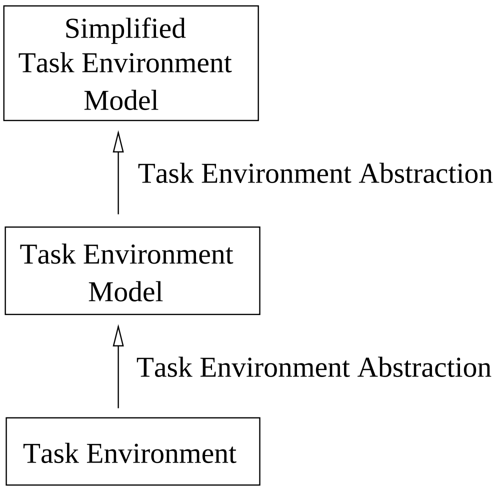
Simplified Task Environment Model
Task Environment Abstraction
Task Environment Model
Task Environment Abstraction
Task Environment

Figure 1.2: An task environment abstraction is a simplification of a task environment or a task environment model.

1.5 Planning Model Types

Task planning is a common way for autonomous agents to figure out what to do to reach a certain goal or to maximize its utility. When task planning is used for execution in real world environments, it is necessary to perform some kind of abstraction to construct a suitable task environment model that can be solved with search or optimization algorithms where the solution is a description of what should be done next. Many different types of such *planning models* exist that can be classified according to the dimensions mentioned in Section 1.2.

Definition 1.3
A ***planning model*** *is a task environment model that models the performance measure and sequential nature of a task environment*

Definition 1.4
A ***planning model type*** *is a set of models of a task environment class*

The term *problem model* or simply *problem* is in this thesis used to denote models that can have solutions but are not neccesarily sequential in nature and/or includes a performance measure. A planning model is therefore considered to be a specialized problem model.

The so called *classical planning* model type makes the strongest assumptions according to the dimensions of task environments. The Planning Domain Definition Language (PDDL) [30] is often used to specify planning models in a succinct way with the help of a logical formalism. Many different versions exist with various levels of expressivity [26] [29].

One extension of PDDL makes it possible to express so called Markov Decision Processes (MDPs) [38] [62]. MDPs can express uncertainties in action outcomes and exogenous events with probability distributions and use rewards to express the planning agent's performance measure. MDPs are currently one of the most commonly used models for planning under uncertainty.

MDPs make the assumption that the environment is fully observable, which is often not an accurate abstraction; it is sometimes necessary to model that an autonomous agent equipped with a camera is unable to see through walls. An extension to the MDP model is the Partially Observable MDP (POMDP) [38] which can model partially observable environments at the expense of increased complexity of solving the specified problems. Approximative solution methods are often used to make it possible to scale up beyond systems containing just a few states [49].

Other types of planning models exist such as *conditional planning* models which often assume that the world is non-deterministic and sometimes also partially observable. The solution to such models are conditional plans that can contain if-then-else constructs and while loops [8] [5].

Another type of planning model type is based on *constraints* which can be used to represent both plans and goals. This type of planning model is often considered very flexible because parts of a plan can be provided by a user which can be elaborated by the planning system. Such a planner can also be used in a mixed-initiative framework where different users add constraints [41] [51] [28]. Constraint-based planners are often very expressive but often good heuristics must be manually constructed to solve large problems and the assumption is often that the task environment is fully observable and deterministic.

In this thesis, a highly expressive graphical model called Dynamic Decision Network (DDN) [14] will be used to represent planning models (see Chapter 3) which can be used to express POMDPs with factored state spaces. A so called *hybrid* DDN can in addition to POMDPs include a mix of discrete and continuous elements, which is very useful when trying to model complex task environments. DDNs will also be used as simulation and evaluation models. It is not possible to solve DDNs exactly so one must rely on approximative solution algorithms and/or use them to construct simplified but solvable planning models.

By looking back at the properties of task environments, one can see that Hybrid DDNs make very few assumptions that limit the expressivity of a task environment. In the case studies (see Chapter 5 and 6), the environment will also be considered to be *dynamic* as well which means that the deliberation time of the agent will be taken into account. The only assumption that will be left untouched is the assumption of a single agent. Several external agents will be modelled but they will not be assumed to be goal-reaching or utility-maximizing.

1.6 Constructing Planning Models

When a planning model (see Section 1.5) is to be constructed for an autonomous agent, it is important to take the properties of the task environment into consideration when the model is supposed to be used for guiding the agent's execution.

The available computational resources must also be taken into account so that it does not take too long to provide a solution. Planning is in general much harder than other tasks such as prediction or state estimation because it is necessary to predict many different state trajectories that are caused by the agent's actions. This means either that the used models must be kept on a high level of abstraction for handling longer temporal horizons [1] or kept rather small to make it feasible.

There is often also a huge number of possible ways to represent actions and sensors in different types of planning models and the most detailed and accurate action or sensor description is not neccesarily the best.

The most common way to construct a model for task planning is that a human user first decides what types of abstractions to perform and why, and then specifies how the actuators, sensors, environment and performance measure will be abstracted and used in the resulting model. This process is considered rather difficult because the resulting model should be both as general as possible to avoid the construction of several planning models, while at the same time take the available computational resources and the requirements for a timely response into consideration. The resulting planning model often becomes a rather coarse view of how the environment works and many times also the *only* view for planning on that level of abstraction.

1.7 Focus of Attention

Due to the manual abstraction process and construction of planning models, agents that are supposed to use these models for planning their execution are typically not provided with any "focus of attention" mechanism where

[1]There is a difference in both temporal and spatial horizons between a model that uses concepts such as "fly to point p" and another that uses "descend 0.3 meters" instead.

they can choose or construct the models themselves that are most appropriate for the situation and task at hand. Such a capability would be very useful in complex task environments where it is not possible to use detailed models for everything that can be relevant *all the time*. However, the agent must still be able to construct or select more detailed models of its task environment when necessary, such as in situations where "something" in the task environment does not behave as in the "normal" case and that more detailed reasoning is required to resolve the problem.

How can agents then be given the ability to focus their attention during planning and execution? In this thesis, agents are given this ability through the use of dynamic planning models which are constructed depending on what effect they will have on the performance during execution. This means that the accuracy of the model (which is often used to evaluate models in general) is allowed to decrease if the performance of the agent increases which makes the model less accurate but also more feasible.

Dynamically changing planning models can be viewed as an instance of the more general problem of selecting or generating *any* simplifying model with some notion of suitability of models that depends on that model type's particular purpose. This problem will be described next.

1.8 Dynamic Abstraction

The term *dynamic abstraction* refers to the capability of a system to dynamically change its simplifications of its task environment or models thereof depending on the current circumstances. Since any system with limited computational resources that needs to operate in and model a complex task environment would have to perform abstractions, the capability of dynamically changing the currently performed abstractions would provide both a more flexible and capable system. A system that can perform dynamic abstraction can choose how its environment should be modelled for the purpose of e.g. knowledge representation [10], prediction, explanation and planning.

For knowledge representation, dynamic abstraction will enable an agent to represent knowledge at many different levels of abstraction and select suitable versions of knowledge to reason with depending on the situation. Humans are believed to be particulary good at this task and seem capable of dynamically changing their view of an environment and constructing and reasoning with abstract concepts.

An agent's prediction and explanation mechanisms could also be improved by using dynamic abstraction since the agent could then dynamically select what variables to take into consideration to get as good result as possible, depending on its own available computational resources. In general, more complex models for prediction and explanation make it more computationally intensive and the important tradeoff between accuracy and feasibility must be balanced at all times.

Dynamic abstraction for task planning, which is the focus of this thesis, will be discussed in more detail in Section 1.9.

The general task of dynamically constructing models for a particular task is not easy. It is not clear how to combine different types of models into one that makes sense and how the resulting models can be evaluated in order to improve the abstraction process. Compositional or component-based modelling techniques [22] seem to make things easier, but the task is far from trivial in the general case. Under limited circumstances though, it is at least currently possible to perform dynamic abstraction for task planning in combination with plan execution, where it is possible to directly evaluate the performance of an agent that uses a certain planning model. More information about the state of the art in dynamic abstraction can be found in Section 1.10.

1.9 Dynamic Abstraction for Planning and Execution

The main topic of this thesis is how dynamic abstraction can be used for planning in the context of execution. The connection to execution is important because feedback from the execution should influence the dynamic abstraction procedure.

Many years of research within task planning have produced many different types of planning models and solution methods. The method used in this thesis is to try to reuse these results.

1.9.1 Example

Imagine an agent that operates in a complex task environment and is able to perform dynamic abstraction in order to construct simplified planning models that captures the most important aspects of its environment. It is assumed that the models can be solved with a suitable solution method. Suppose that the generated planning models are instances of MDPs (See Section 1.5 and Chapter 3) and that the agent is able to reason about its computational resources in order to keep the models on a suitable level of abstraction that enables the solution method to provide a solution within a reasonable time. An example of how the actual generation of MDPs can be performed with e.g. continuous task environment models is discussed in Chapter 5.

The agent then solves the planning model in order to use the solution during execution. For MDPs, the solution is a so called *policy* which map all possible discernible states to an action that should be executed in the corresponding state. But the states and actions have been dynamically constructed by an abstraction technique so we have to discuss further what could possibly happen when the agent should execute the solution.

First of all, unexpected or simply ignored events in the agent's environment might cause the problem model used during the solution phase to become invalid or unsuitable after a while. The agent might for example discover that an object, previously assumed to be a stationary obstacle, is in fact a vehicle that it is supposed to inspect. The agent may have to change its way to perform abstractions and replan if (or perhaps more accurately, *when*) this happens. It would be beneficial for the agent that discovered the vehicle if it is capable of changing its way to view its environment by taking the speed and direction of the vehicle into consideration in order to predict where it is going. One may then argue that the agent should have used that view from the very beginning, but then one also must consider that the agent both has limited sensor capabilities and computational resources. It is therefore just not feasible for the agent to represent everything in the most detailed manner just because something might turn out to be more complicated than previously perceived. Remember that the agent has made a decision about its abstractions and thereby focused its attention on the parts of the environment that it considered to be most important. It has made an effort to make a good tradeoff between accuracy and feasibility of the planning model.

Another problem is that it might turn out that a solution to the MDP might be on such a high level of abstraction that the agent is incapable of executing it. This depends of course on how complex the agent's available behaviors or skills (see Section 2.5.3) are but the agent is assumed not to have skills for everything because then it would not need to plan at all. Our UAS uses an execution system where parameterized reactive skills such as "Fly to a point p", "Take off" or "Turn camera towards point p" exist but more complicated missions like "Deliver a set of packages to a set of destinations" does not have a direct match to such a skill and task planning techniques are used instead. Complicated missions need to be planned down to the level where skills are available to carry out the solution. The point is that an abstract solution might need to be refined somehow, which is a common and natural technique used within Hierarchical Task Network (HTN) planning [21] and Hierarchical Reinforcement Learning [6]. For an MDP policy, each planned action might expand into a subproblem[2] of its own which can also be constructed with the agent's dynamic abstraction capabilities. These resulting subproblems, which could be MDPs or other planning model types need to be solved as well. The refinement should stop when there are skills available that can reliably enough execute at least parts of a solution.

The refinement of solutions in this manner creates an abstraction hierarchy which the agent needs to keep track of and check if they are still valid and suitable during execution. If higher level problem models suddenly become too inaccurate due to ignored, simplified or changing conditions, there

[2]The term *problem* is used here because it might be the case that an episodic model is used.

is a risk that the lower level solutions might be invalid or irrelevant.

1.9.2 DARE

All these consequences of using dynamic planning models have been studied in this thesis where a method called DARE (stands for Dynamic Abstraction-driven Replanning and Execution) has been developed that tries to handle these problems (see Chapter 4). DARE is a very abstract method and needs to be instantiated with a particular task environment class before it can be used. Chapter 5 presents an instantiation of the DARE method where MDPs are used as the planning model type and subproblems are constructed dynamically and solved until a sufficiently detailed level of abstraction is reached. In Chapter 6 DDNs are constructed instead which makes it possible to represent partial observability.

1.10 Related Work

The general idea that abstractions are necessery for decision-making is certainly not a new one. The development of HTN-planning [71] [64] was largely driven by the need for planning with more abstract plan operators first, forming an abstract plan, and then refining the solution and backtracking if necessary. The SIPE planning system [75] was one of the first domain-independent HTN-planners which was described in great detail. In the book that describes SIPE [75], there was a discussion about stopping the refinement of task networks and only constructing plans at a certain level of abstraction. This idea was implemented in CYPRESS [77] where the so called Act language [76] was used to describe both planning and task execution refinement in the same language. Planning was only performed down to a certain level and then the task refinement kicked in with the help of the PRS [52] execution system.

Other domain-independent HTN planning systems such as O-Plan [72] and SHOP2 [53] have been developed for real-world applications and the current situation is that when a planning system is used for solving large and real-world like task planning models that require reasoning on several levels of abstraction, HTN-planners with added capabilities of dealing with many different types of constraints are often used.

Automatic abstraction has also been developed for so called STRIPS or classical planning [31] domains. A system called ALPINE [40] was used to automatically generate abstraction hierarchies given a domain description.

Most of the work within dynamic abstraction for stochastic task environments has been done within the area of hierarchical reinforcement learning (HRL) [6] which can be viewed as a generalization effort for HTN-planning in stochastic environments. The main idea is to use abstract MDPs that can use sub MDPs almost like primitive actions. There are many ways to actually do this but the three most cited HRL systems are the Options framework [70], MAXQ [16] and Hierachical Abstract Machines (HAM) [60].

Jonsson and Barto [37] used a modified version of the U-Tree algorithm [46] to automatically find state abstractions in the Options framework. A U-Tree is a form of decision tree which keeps track of the state abstraction by using a statistical test to select when distinctions between different states should be made.

Hengst [35] has developed an algorithm called HEXQ which learns subtask structures by separating state variables that change at different rates.

Mannor et al. [45] use clustering techniques to perform dynamic abstraction by looking at the state transition history which is converted to a graph where the clustering takes place. The clusters are then used to learn policies that move between the different clusters which can then be used as abstract actions.

Steinkraus and Kaelbling [66] use a structure very similar to the HSN-structure (see Section 4.5), where different abstractions are performed on the way down to the most detailed abstraction.

Another piece of work that is very much related to this thesis is [43] where the authors try to dynamically generate models depending on the questions asked about a certain system. They express preferred models with a theory of abstractions about model fragments.

Some work with hierarchical POMDPs has been done as well ([73] contains a small survey) but not to automatically select abstractions for POMDPs at different levels. Kaelbling et al. [73] use reinforcement learning methods to learn abstract policies over macro actions and demonstrate many benefits of using abstract states in POMDPs.

Sturtevant and Buro [67] uses abstractions in the context of path planning and path execution where the original path planning model is transformed into a hierarchy of models on different levels of abstraction. The abstraction is performed by looking for cliques in the planning graph or tiles.

1.11 Contributions

Although the idea of dynamic abstraction has been used within hierarchical reinforcement learning, it has never been approached from a more general point of view where *different types* of planning models are combined and selected or generated depending on the current situation and task. The DARE method tries to find a way to do this in a more general framework and uses the connection to skills that are supposed to execute the solution. Figure 4.2 on page 42 illustrates a vision where many different types of planning models are combined and updated dynamically.

Another contribution is the idea and implementation of using the task environment's performance measure to focus the attention when the planning models are constructed. In Chapter 5, this is done with the use of relevance functions. Chapter 6 uses a measure of the expected utility of points to build planning models.

The two case studies contribute to the area of dynamic abstraction by demonstrating that (parts of) the abstraction process can be formulated as solutions to optimization problems.

The results presented in the case studies have been published in [57] and [58].

1.12 Outline

The rest of the thesis is organized as follows: Chapter 2 briefly provides preliminary information about probability theory, Dynamic Bayesian Networks (DBNs) and local optimization techniques. Chapter 3 describes a general graphical model called Dynamic Decision Networks (DDN) which can be used to model stochastic and partially observable planning models. DDNs are used to simulate the task environments used in the case studies. The same chapter also describes MDPs as a special case of DDNs where all variables are discrete and observable.

Chapter 4 describes the DARE method which tries to pinpoint the necessary capabilities of a planning and execution agent that uses dynamically generated planning models and needs to keep track of the corresponding abstractions' validity.

Chapter 5 presents the first implementation of DARE where the task environment class is fully observable. The planning is performed on several levels of abstraction and the planning models (MDPs) are dynamically generated according to the currently best considered abstraction. Chapter 6 contains a description of the second case study where planning models are dynamically created in a partially observable task environment class. In this implementation the levels of abstraction are fixed but the planning models are still generated dynamically depending on the agent's current belief state.

Finally, Chapter 7 contains some conclusions and descriptions of future work.

Chapter 2

Preliminaries

Since this thesis is concerned with how suitable planning models can be generated depending on the current circumstances, it is useful to know about some of the tools to construct models that are used in the case studies in Chapter 5 and 6.

Probability theory is often used to model uncertainty. In this thesis, probability theory is used to specify task environment models and is also the basis of filtering algorithms in partially observable environment classes. A brief introduction to the relevant concepts in probability theory will therefore be given in Section 2.1.

The graphical model called *Bayesian Networks* (BNs) will be used to succinctly specify probability distributions and illustrate dependencies between variables in the stochastic planning models used in the case studies. The temporal version of BNs, Dynamic Bayesian Networks (DBNs), can be used to describe stochastic processes and is the basis for the Dynamic Decision Networks (DDNs) [14] that will be described in more detail in Chapter 3. BNs and DBNs are described in Section 2.2 and 2.3.

The introductory chapter contained a description of the tradeoff between accuracy and feasibility of a certain model. When tradeoffs are performed by computers, they are often formulated as an *optimization problem* and solved with some of the available techniques. This is also the case in this thesis and a brief introduction to optimization problems and relevant solution techniques is provided in Section 2.4.

A solution to a planning model should be something that can be executed by an execution system, and since this thesis discusses the important connection between planning and execution, an introduction to reactive execution systems in general and the Modular Task Architecture (MTA) (described in [56]) in particular is given in Section 2.5. MTA is used in the UAS Tech software architecture to structure the execution and it uses the Common Object Request Broker Architecture (CORBA) [59] for communication.

2.1 Probability Theory

The concept of *variation* is often a central part of many applications. Variation in this context means that the outcome of some event such as tossing a coin or using a sensor, varies even if the initial conditions are perceived to be the same.

The cause of the variation can be discussed and it might be the case that the phenomenon of study is actually deterministic if all the variables are taken into account. The problem is that it is not always possible to get access to all the variables that determine an outcome (the environment might be partially observable) and therefore it is necessary to deal with variation, whether the world is deterministic or not.

Probability theory is one way of representing variation and it is used in many practical situations such as representing measure error and building stochastic models such as MDPs (see Chapter 3) used for task planning.

2.1.1 Basic Assumptions

The basic assumption of probability theory is that there exists a universe of outcomes U and each *event* $E \subset U$ (given some basic assumptions about E such as it must be a σ-algebra [20]) is given a number between 0 and 1, called the probability $P(E)$ of the event E. The probability function P is constrained by the following fundamental axioms of probability:

- For any set $E \subset U$, $P(E) \geq 0$
- $P(U) = 1$
- Any countable sequence of pairwise disjoint events $[E_1, E_2, ...]$ satisfies: $P(E_1 \cup E_2 \cup ...) = \sum P(E_i)$

2.1.2 Stochastic Variables

Stochastic variables are often used to specify events which can be denoted by expressions such as "$Alt_{\text{UAS}} > 10.2$" [1] whose denotation defines the event where the UAS's altitude is above 10.2 meters. The stochastic variable Alt_{UAS} is in this case used to represent the altitude. Events, such as the one just mentioned, can be given probabilities as long as they follow the fundamental axioms of probability. Alt_{UAS} is an example of a stochastic variable with a continuous domain (the altitude). Domains can also be discrete sets such as $\{rs_1, rs_2, rs_3\}$, which in this case represents three different road segments in a road network. The expression "$RS_{car} = rs_2$" can

[1]A more theoretical and complete treatment of probability theory (see [20]) define stochastic variables in a different way, but for the purpose of this thesis, thinking about stochastic variables as something that can be used to form expressions that denote events are sufficient

then represent the event that a certain car is travelling on the road segment denoted by rs_2.

Simple expressions such as "$Alt_{\mathrm{UAS}} > 10.2$" can be used to form combined events with logical operations like "$Alt_{\mathrm{UAS}} > 10.2 \wedge Alt_{\mathrm{UAS}} < 20$" which denotes the intersection of the events denoted by "$Alt_{\mathrm{UAS}} > 10.2$" and "$Alt_{\mathrm{UAS}} < 20$", whose intended meaning is that the UAS's altitude is between 10.2 and 20 meters.

2.1.3 Distributions and Density Functions

A discrete stochastic variable X has a so called *probability distribution* associated with it, which defines the probability $P(X = d)$ for all the elements $d \in D_X$ in that variable's domain D_X [2]. A probability distribution for RS_{car} might be $\langle 0.1, 0.7, 0.2\rangle$ which means that $P(RS_{car} = rs_1) = 0.1$, $P(RS_{car} = rs_2) = 0.7$ and $P(RS_{car} = rs_3) = 0.2$. The sum of all probabilities in the distribution must be equal to 1, according to the fundamental axioms of probability. For a continuous stochastic variable, it is not possible to enumerate all possible events and one has to associate a *probability density function* f_X with the variable X instead. Figure 2.1 illustrates an example of such a density function for the Alt_{UAS} stochastic variable. The integral of a probability density function f_X, $\int_{-\infty}^{\infty} f_X(x)dx$, must be equal to 1.

2.1.4 Joint Distributions

Events that involve more than one stochastic variable can be specified with expressions such as "$RS_{Truck} = rs_1 \wedge RS_{Car} = rs_2$" where the stochastic variable RS_{Truck} represents the possible road segments for a truck and has the same domain as RS_{Car}. The probability distribution for both of the two stochastic variables must be defined for every combination of values such as $P(RS_{Truck} = rs_1 \wedge RS_{Car} = rs_1) = 0.1$, $P(RS_{Truck} = rs_2 \wedge RS_{Car} = rs_1) = 0.12$ and so on. The complete specification of all the stochastic variables' probability distributions and density functions is called the *joint* probability distribution. If discrete and continuous stochastic variables are mixed in the same model, the probability density functions for all the continuous variables must be defined for all combinations of values for the discrete variables in the general case.

2.1.5 Conditional Distributions

The *conditional probability* $P(E_1|E_2)$ given two events E_1 and E_2 is defined as:

[2]The probability function P is in this way also used for expressions involving stochastic variables and the intended meaning is the probability of the *denoted* event.

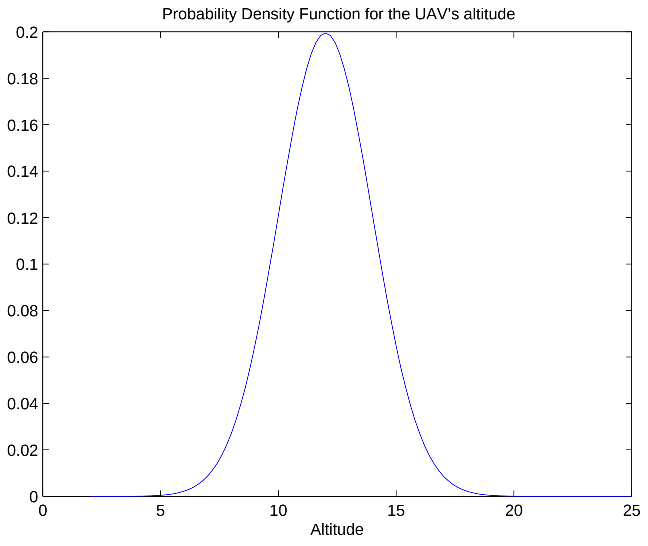

Figure 2.1: An example of a probability density function associated with the Alt_{UAS} stochastic variable.

$$P(E_1|E_2) = \frac{P(E_1, E_2)}{P(E_2)} \tag{2.1}$$

and is often used to model partially observable events or state transition distributions in e.g. MDPs.

The *conditional probability distribution* $\mathbf{P}(X|Y)$ for the two discrete stochastic variables X and Y is defined as:

$$\mathbf{P}(X|Y) = \frac{\mathbf{P}(X, Y)}{\mathbf{P}(Y)} \tag{2.2}$$

Equation 2.2 should be interpreted as the set of equations:

$$P(X = x_i|Y = y_j) = \frac{P(X = x_i \wedge Y = y_j)}{P(Y = y_j)} \tag{2.3}$$

for all combinations of the stochastic variables' domain elements x_i and y_j.

Similarily, the *conditional density function* $f_{X,Y}$ for two continuous stochastic variables X and Y is defined as:

$$f_{X,Y}(x|y) = \frac{f_{X,Y}(x, y)}{f_Y(y)} \tag{2.4}$$

where $f_{X,Y}$ is the joint probability density function for X and Y and f_Y is the (marginalized) probability density function for Y which is equal to $\int_{-\infty}^{\infty} f_{X,Y}(x,y)dx$.

2.1.6 Bayes Rule

Bayes Rule is useful when one needs to calculate the probability $P(E_i|E_j)$ in terms of $P(E_j|E_i)$, $P(E_j)$ and $P(E_i)$:

$$P(E_i|E_j) = \frac{P(E_j|E_i)P(E_i)}{P(E_j)} \tag{2.5}$$

Bayes rule is used extensively in probabilistic expert systems because it is often the case that it is difficult and/or inappropriate to estimate or measure the conditional probability in a certain "direction" but not the other way around. In Chapter 6, Equation 2.5 will be used to calculate the probability of a state given noisy sensor data.

Bayes rule can be extended to hold for distributions and density functions similar to Equation 2.2 and 2.4.

2.1.7 Expectation

For stochastic variables that have a domain of a subset of the integers or reals, it is possible to define the expected value given the corresponding distribution or density function. The expectation of a discrete stochastic variable X is defined as follows:

$$E(X) = \sum_{x \in D_X} xP(x) \tag{2.6}$$

The corresponding expression for continuous stochastic variables involves an integral instead of a sum:

$$E(X) = \int_{-\infty}^{\infty} x f_X(x)dx \tag{2.7}$$

The expectation is often denoted m_X and generalizes to vectors of stochastic variables $\mathbf{X}$ as well.

2.2 Bayesian Networks

One of the major problems with probabilistic models is that the number of probabilities that must be specified for the joint distribution grows exponentially with the number of variables in the general case.

There are often more succinct ways of implicitly describing a probability distribution by only specifying conditional distributions between the variables in the model. A graphical model called *Bayesian Networks* provides this functionality and has been credited with the extensive use of probabilistic techniques in AI applications because of the resulting increased efficiency of probabilistic reasoning for larger models.

2.2.1 Definition

A Bayesian network is a Directed Acyclic Graph (DAG) where each node represents a stochastic variable. Arcs between nodes should as closely as possible represent "direct" causal influences between variables. Figure 2.2 illustrates a typical Bayesian Network.

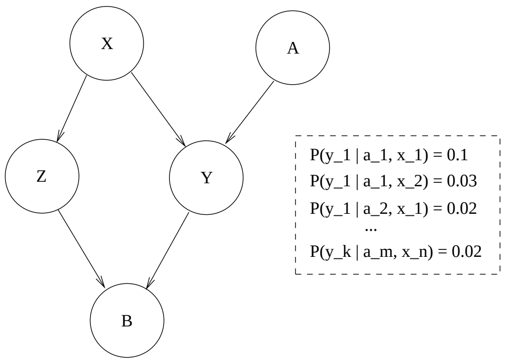

Figure 2.2: A typical Bayesian Network. Parts of the conditional distribution for the stochastic variable Y, given its parents values, is also shown.

Instead of specifying the full joint distribution for a set of stochastic variables, one only needs to specify the conditional probability distributions or densities of variables given their parents.

The main assumption for a Bayesian Network with discrete variables $X_1, ..., X_N$ is that the full joint distribution $P(X_1, X_2, ..., X_N)$ can be calculated by:

$$P(X_1, X_2, ..., X_N) = \prod_{i=1}^{N} P(X_i|\text{Parents}(X_i)) \qquad (2.8)$$

where $\text{Parents}(X_i)$ is the set of parent nodes of X_i. Figure 2.2 illustrates parts of one of these conditional probabilities $P(Y|X, A)$ which is a lot easier to specify than the full joint distribution $P(A, B, X, Y, Z)$ for all combinations of the variables' domains.

2.2.2 Hybrid Models

So called *Hybrid* Bayesian Networks can include both discrete and continuous stochastic variables. A continuous variable with discrete and continuous parents must then have several conditional density functions for each combination of discrete values that may depend on the values of the continuous parents as well. A discrete variable with continuous and discrete parents must in the general case have several density functions defined. Hybrid BNs will be used to define parts of the task environment models used in the case studies (Chapter 5 and 6).

2.2.3 Inference

BNs can be used for many different purposes. One of the most basic capabilities is to calculate a conditional probability $P(E_i|E_j)$. Events described with stochastic variables are commonly used to formulate such *queries* where it is often the case that a set of discrete variables $\mathbf{Y}$ are already known and another set of discrete variables $\mathbf{Z}$ are unknown. The probability distribution $\mathbf{P}(X|\mathbf{Y},\mathbf{Z})$ of the query variable $\mathbf{X}$ is then calculated by the inference procedure applied to the BN.

In this thesis, inference in Bayesian Networks is performed in a specialized context of *filtering* where the probability distribution over the *current* state is estimated when a system is described with a so called Dynamic Bayesian Network (see Section 2.3 and 6.4 for more information about this particular form of inference).

2.2.4 Implicit Models

All the examples given so far have used either a tabular or explicit probability density representation of the probability distributions. This is not always possible or even necessary. Suppose that one would like to represent the probability distribution of the values of a laser scan sensor given the sensor's pose and a map of the surroundings. An explicit probability density representation of the sensor values given all possible poses is not possible to store due to the continuous stochastic variables. It is possible to calculate one such density *given* that the pose is known, by using raycasting or similar techniques in the map, which is an example of an *implicit* representation of a density function widely used in practice [74], e.g. when a mobile robot performs Monte Carlo Localization (MCL) [25] with a particle filter (which is an approximative inference method that does not need an explicit density representation).

Implicit densitiy functions will be used in the case studies e.g. when visibility conditions are used. Such conditions are very cumbersome to represent explicitly but relatively easy to calculate with raycasting techniques.

2.2.5 Model Estimation

A conditional distribution can be *estimated* by a suitable statistical technique such as general function approximation (if no particular type of distribution is assumed) or parameterized models like Gaussian distributions or mixture models. Maximum Likelihood (ML) (see [2]) parameter estimation is a common method which is used in Chapter 5 to estimate transition distributions in MDPs.

2.3 Dynamic Bayesian Networks

Bayesian Networks are a great tool for modelling situations where temporal aspects are not taken into consideration. It is often the case that one would like to model stochastic systems that evolve over time as well, and then an extension to Bayesian Networks can be made to create *Dynamic* Bayesian Networks (DBNs) [13].

DBNs can be used to model stationary stochastic processes which makes the so called *Markov assumption*. Temporal stochastic models that make this assumption assume that any state can only depend on a finite history of stochastic variables. In a *first order* Markov model the current state can only depend on the previous one, which is often the case for DBNs. A first order Markov model can represent any finite order Markov model by introducing extra stochastic variables.

Figure 2.3 shows the typical structure of a DBN model. The basic idea is to use a set of stochastic variables for each time step and define the conditional probabilities of the variable set at time t given the variable set at time $t-1$. The Markov assumption makes it possible to define a DBN with only two sets of variables, one set of prior distributions for time step 0 and a set of conditional distributions for each stochastic variable.

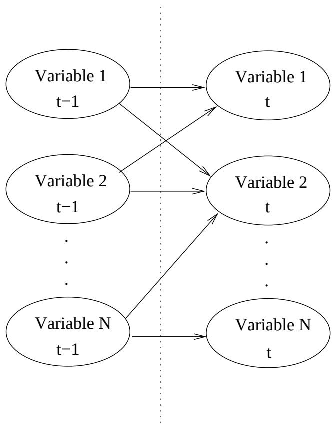

Figure 2.3: A typical Dynamic Bayesian Network.

Given that a DBN can represent a probability distribution over time, it can be used by an autonomous agent to perform the following types of reasoning over time:

- The probability distribution over the agent's current state space can be updated with *filtering*. This is a very useful operation when the task environment is partially observable.
- Future distributions can be estimated with *prediction*. This operation is useful when the agent performs planning.
- The set of *previous* state distributions can often be better estimated when more information is received which is called *smoothing* and sometimes *hindsight*. Smoothing is especially useful if any type of machine learning [48] is used to update the distributions in the DBN.

Filtering is used to compute the posterior probability distribution $\mathbf{P}(X_{t+1}|Y_{1:t+1})$ where X_{t+1} and $Y_{1:t+1}$ represents all unobserved variables at time $t+1$ and the sequence of observed variable sets $(Y_1, Y_2, ..., Y_{t+1})$ respectively.

The filtering computation can be described with the following equation:

$$\mathbf{P}(X_{t+1}|Y_{1:t+1}) = \frac{\mathbf{P}(Y_{t+1}|X_{t+1}) \sum_{x_t} \mathbf{P}(X_{t+1}|x_t) P(x_t|Y_{1:t})}{\mathbf{P}(Y_{t+1}|Y_{1:t})} \tag{2.9}$$

where $\mathbf{P}(Y_{t+1}|X_{t+1})$ often is called the *observation model* which treats the stochastic variables Y_{t+1} as observations of the hidden variables X_{t+1}. $\mathbf{P}(X_{t+1}|X_t)$ is often called the *transition model*.

The corresponding equation for performing prediction $k+1$ steps into the future with a DBN is the following:

$$\mathbf{P}(X_{t+k+1}|Y_{1:t}) = \sum_{x_{t+k}} \mathbf{P}(X_{t+k+1}|x_{t+k}) P(x_{t+k}|Y_{1:t}) \tag{2.10}$$

A special case of DBN is the *Markov chain* where only one discrete stochastic variable with finite domain is allowed. It is possible to model any discrete DBN with a single discrete variable, but it is often much easier to use several variables since the domain can often be described more succinctly with that approach. Markov Decision Processes (MDPs) (see Section 3.4) use Markov chains to model a stochastic system that is fully observable.

2.4 Optimization

Many planning model types make it possible to specify some kind of cost or utility of a solution to the problem. There are sometimes many different ways to solve a specific planning model but some solutions might be better than others due to different circumstances. There might for example exist many different paths for a UAS to fly from one point to another but some paths might be better than others due to the length of the path or other criteria. When there are several possible solutions available to a model and the solutions can be compared in terms of cost or other measurements, the model is called an *optimization problem*. In this thesis, optimization problems are used for other tasks than planning and a solution to such a problem will typically not be a plan or an action but instead a decision about abstraction parameters.

One possible definition of an optimization problem is that there exists a set of variables V where each variable has a domain D_V that can be any type of set. Let D be the crossproduct of all the variables' domains $D_{v_1} \times D_{v_2} \times ... \times D_{v_{|V|}}$. The objective function $U : D \to \mathbb{R}$ is then used to compare different solutions.

The form of D and U specifies the possible solution methods that can be used. When the set of variables all have discrete domains, a common method is Branch and Bound [42] or some local search method such as hillclimbing and simulated annealing [39]. For pure continuous domains and linear utility functions, Linear Programming [65] can be used. Other optimization algorithms make use of the *gradient* and *hessian* of U to guide the search towards a global or local maximum [55].

In this thesis, different types of local search are used to perform optimization as a method to select abstractions. Local search methods make use of a so called *neighbourhood* function $N : D \to 2^D$ which defines the possible successors when the local search method is exploring a certain state $d \in D$.

N can be used in different ways to perform the search. In hillclimbing search, all the states in the neighbourhood are examined and the one with the best utility is selected and set as the new current state. The neighbourhood function can also be sampled, as in simulated annealing, where the current state is set to the sampled state with a certain probability that depends on the utility and the so called *cooling schedule* which makes choices that are worse, less probable with time.

2.5 Execution Systems

There are many possible ways to structure execution in an autonomous agent that may have to operate in dynamic task environments with both action and sensor uncertainty. Researchers within the area of *AI robotics* [50] have been dealing with these isssues and different paradigms for structuring execution have evolved. The evolution started with Shakey [54] and the so called *hierarchical paradigm*, which is often very computationally expensive and makes use of task environment models, and continued with the *reactive paradigm* with behaviors with only simple memoryless modules. The hybrid *deliberative reactive paradigm* is now more commonly used which tries to combine the use of models with behaviors.

2.5.1 Modular Task Architecture

The UAS Tech system uses the Modular Task Architecture (MTA) (see [56] for a description) to structure much of the execution and is classified as a hybrid deliberative reactive architecture. MTA makes use of the Common Object Request Broker Architecture (CORBA) [59] for communication. The parameterized behavioral components are called Task Procedures (TPs). TPs have a standardized way to be initialized, terminated and so on and can call any CORBA service. In the UAS Tech system, a path planner, a geographic information system and knowledge processing middleware DyKnow [19] are all accessed as CORBA services. Before any action can be executed, a TP instance (TPI) must be created by a request to a TP library service.

TPs have a behavioral component that specifies the execution. Currently the implementation of the behavioral component is based on state machines and the TPs are allowed to have local and service reference variables.

A TPI can create other TPIs as well and structures of TPIs can be constructed. Figure 2.4 illustrates a set of TPIs that is used in a part of a mission where a set of building structures are supposed to be investigated. In the UAS Tech system, the set of TPIs are changing all the time. The set of instances during takeoff or landing is different from the ones shown in Figure 2.4.

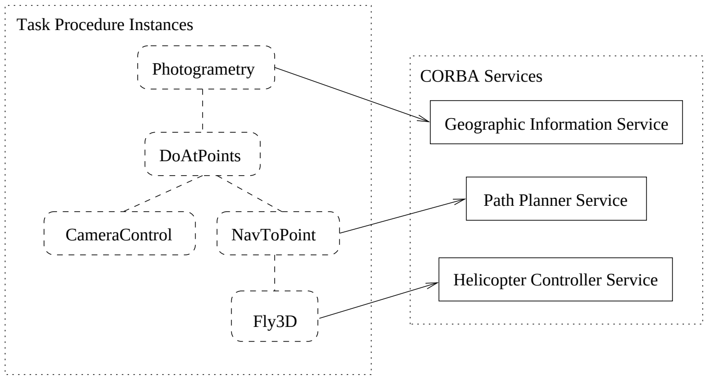

Figure 2.4: Task Procedure instances for the part of a mission where a set of building structures are investigated. The figure also shows the services that the TPIs are using.

2.5.2 Other Architectures

Many execution system architectures have been developed which have many similarities with MTA due to the use of some notion of task or task instance that can be used to perform execution. The Reactive Action Package (RAP) [23] system uses hierarchically structured "packages" that can perform simple tasks in different ways depending on the situation. The packages can start up other packages to perform subtasks and the concurrently executing tasks are managed with the RAP memory and a mechanism for synchronization. The Procedural Reasoning System (PRS) [52] with successors (e.g. Apex [27]) builds on similar ideas. PRS uses a simple database with facts that can be filled in by sensors and task executing behaviors.

2.5.3 Definition of Skills

Since there are a lot of similarities between different execution system architectures such as MTA, RAP and PRS, the collective term *Skill* will be used to refer to any primitive task executing piece e.g. TPIs, RAPS and so on:

Definition 2.1
*A **skill** is a reactive computational mechanism that achieves a certain objective in a limited set of circumstances with the following capabilities:*

- *It can have an internal state and/or use a shared database.*
- *It can send messages to other skills either through an event system or a database.*
- *It is possible to control and monitor it from the "outside" and terminate it safely on demand, even if it is not finished with its objective.*

These capabilities of skills are used to keep the DARE method (see Chapter 4) as general as possible and not specialize it too much towards use of MTA. MTA, RAP and PRS are all capable of defining skills that have these properties.

Chapter 3

Dynamic Decision Networks

The previous chapter described the Dynamic Bayesian Networks (DBNs) that can be used to model any discrete time stochastic process that satisfies the Markov assumption. DBNs can be extended to specify planning models and are then called *Dynamic Decision Networks* [14] (DDNs) which are based on *Decision theory* [7]. Decision theory is a very general method for decision-making under uncertainty.

The main idea is to add action and reward nodes to the basic DBN to make it possible for an agent to control the process and determine the utility of possible outcomes.

This chapter also describes the Markov Decision Process (MDP) [62], which can be viewed as a special case of DDN. Partially Observable MDPs (POMDPs) [38] are also a special case of DDNs but are not presented in this chapter because none of the specialized POMDP solution techniques are used in this thesis.

3.1 Example

Figure 3.2 shows an example DDN which describes the problem of delivering a box to a target position on the ground. The box is attached to a winch on a UAS (see Figure 3.1). It is a standard (but often difficult) task of mathematical model building to create a model of this system using the laws of physics combined with parameter estimation techniques and possibly a collision detection/handling system to predict the movement of the box if it touches or falls on the ground.

In order to control the system, it is useful to define a performance measure model. DDNs try to solve this modelling problem by defining a continuous *reward* variable R_t that specifies the immediate reward when the system goes from one state to another ($t-1$ to t). The main objective is to maximize the total reward, $\sum_{t=t_0}^{T+t_0} R_t$, from the current state X_{t_0} with a finite horizon T, but many variants exists such as maximizing the average reward or using a *discount factor* with an infinite horizon (see Section 3.2). The reward variable can in this example e.g. depend on whether the box is "damaged" by colliding too fast with the ground and the behavior of the UAS (if it is too close to the ground or makes some unwanted manouver) etc. Goals, such as "the box should stand on the ground and be detached from the winch", can be specified with a large reward when the box is close enough to the target position with zero velocity while at the same time being detached.

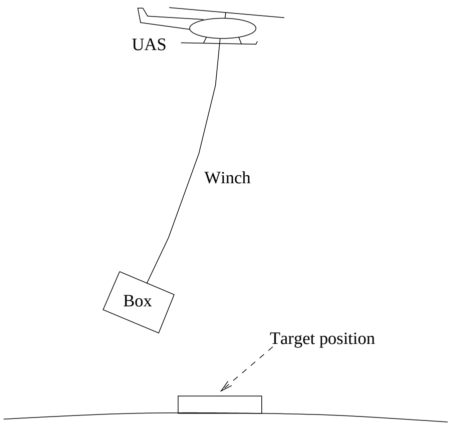

Figure 3.1: A UAS with a winch, trying to deliver a box.

Figure 3.2 only specifies the relationship between different classes of random variables and the contents of e.g. **UAS State** is not specified. Many different versions are possible such as using discrete domains for all variables, which makes it possible to directly model the problem with a POMDP, or try to model most of the relationships with linear Gaussian distributions to make it possible to use feasible filtering and prediction techniques.

The reward variable can be used to specify the desired behavior of the system. It is necessary to make a tradeoff between the UAS's and the box's safety which raises important questions such as: How bad is it to crash compared to a destroyed box? What is best, a solution that takes one hour that succeeds flawlessly with probability 0.99 or a solution that takes five minutes and succeeds with probability 0.95?

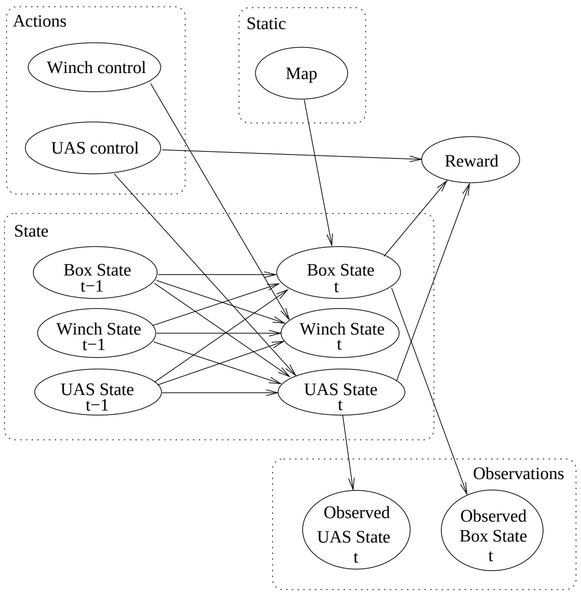

Figure 3.2: A DDN that describes the sequential decision problem of delivering a box.

3.2 Local Reward, Global Utility

The local reward variable in DDNs can be used to specify a *global utility* function U of all states. This utility function typically depends on the sum of the rewards. With U given with fully observable state variables, an agent that tries to maximize its utility can simply perform a one-step application of the $\mathbf{P}(X_t|X_{t-1}, A_t)$, where A_t is the set of action variables, to maximize the *expected utility* (EU) by selecting the action a that satisfies:

$$\arg\max_{a\in A_t} \sum_i U(x_t^i)P(x_t^i|X_{t-1}, a) \tag{3.1}$$

This is a very general principle of decision-making under uncertainty and central in the area of *Decision Theory* (DT) [7].

When the state is only partially observable, the utility function typically depends on probability distributions over states instead which makes the problem of both representing U and finding a solution much more complex.

The global utility of a state can be used when the decision horizon T is finite. When the horizon is infinite, the most common method to make it possible to compare infinite sequences of rewards is to assign a so called *discount factor* $\gamma \in [0..1)$. The discount factor makes sure that all state utilities are finite and it is then possible to compare state trajectories.

3.3 Solution Techniques

A hybrid DDN is generally not possible to solve, with any reasonable definition of solution, due to the simple fact that it can be used to specify any non-linear continuous stochastic Markov model which includes POMDPs and many optimal control problems as special cases.

A common approximate solution method is to discretize the action nodes (if they do not already have discrete domains) and perform depth-limited lookahead from the current state or (approximated) belief state. It is common that the global utility for the state at the maximum depth is estimated by a heuristic function. It is also common to use iterative deepening depth-first lookahead to make sure that some kind of solution is ready as quickly as possible. The requirements of this type of incomplete search is that the agent must perform the lookahead before every decision is made, which can be computationally intensive, especially if the filtering is expensive.

In Chapter 6, depth-limited lookahead is used in combination with particle filters to make an agent decide what to do when the environment is represented with a Hybrid DDN.

3.4 Special Case: Markov Decision Processes

A special case of DDNs is the Markov Decision Process (MDP) [62]. In this model, all random variables are observable, which means that no filtering is necessary. All variables, except the reward variable, are also assumed to have discrete domains.

To be able to use the common notation for MDPs, a random variable S is defined which has a domain of the cross product of all random variables' domains. An action variable A is defined in the same way for all action variables present in the DDN model. Figure 3.3 illustrates the resulting DDN when the action and state space have been created.

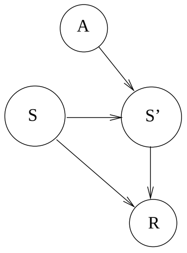

Figure 3.3: A DDN that represents an MDP.

The so called *transition distribution* $\mathbf{P}(s'|s,a)$ defines the probability of ending up in a state $s' \in S$ after executing an action $a \in A$ in $s \in S$.

The *reward density function* $f_R(r|s,a)$ similarly defines the distribution of the reward r received in the same context. $R(s,a)$ is used to denote the expectation of the reward density function when a is executed in state s.

3.4.1 Policy

The assumption of sum of rewards together with the Markov property, makes it possible to define a solution to an MDP as a mapping from the current state to an action [62]. Such a mapping is called a *policy*. It is possible to compute the value V^π of a policy π for all states $s \in S$ through the following recursive formula:

$$V^\pi(s) = \gamma \sum_{s'} P(s'|s,\pi(s))V^\pi(s') + R(s,\pi(s)) \tag{3.2}$$

where s and s' are equivalent to S_{t-1} and S_t respectively. The value is guaranteed to be finite due to the discount factor $\gamma \in [0,1)$ and when $R(s,a)$ is bounded.

3.4.2 Solutions and Solver Methods

Value functions define a partial ordering over policies. A policy π is strictly better than another policy π' if $V^\pi(s) > V^{\pi'}(s)$ for all states $s \in S$. There is always at least one such policy π^* that is strictly better than or as good as all other policies. Such policies are called *solutions* to the MPD.

The value function V^* for all optimal policies satisfies the following *Bellman equation* for all states $s \in S$:

$$V^*(s) = \max_a \sum_{s'} \gamma P(s'|s,a)V^*(s') + R(s,a) \tag{3.3}$$

If the optimal value V^* is known, without any reference to a specific policy, an optimal policy can be extracted from V^* through the following formula:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s') + R(s,a) \tag{3.4}$$

3.4.3 Value Iteration

The Bellman equation can be used to develop an iterative solution algorithm that updates a better and better estimate of the value function V. One such method is called the Value Iteration algorithm [62] and is shown in Procedure 3.1. It uses the so called *Bellman update rule* to iteratively update the value function until the optimality criterium is reached. The optimality criterium is defined by ϵ which means that the resulting policy

π^*_ϵ has a value $V^{\pi^*_\epsilon}$ that may be up to ϵ less than the optimal policy's value V^{π^*}.

Procedure 3.1 VALUE ITERATION(γ)

```
Initialize $V$ arbitrarily, e.g. $V(s) = 0$ for all $s \in S$
repeat
  for all $s \in S$ do
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\gamma V(s') + R(s,a)$
    $\delta \leftarrow \max(\delta, |v - V(s)|)$
  end for
until $\delta < \epsilon(1-\gamma)/2\gamma$
$\pi^*_\epsilon(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)\gamma V(s') + R(s,a)$
Return $\pi^*_\epsilon$
```

3.4.4 Reinforcement Learning

The Value Iteration method described in Section 3.4.3 is quite efficient in its pure form but has some very high requirements. Both the transition distribution $P(s'|s,a)$ and the expectation of the reward distribution $R(s,a)$ must be known in advance, which can sometimes be difficult to calculate in some domains.

There exists a set of MDP solution methods that does not directly require a transition and reward model of the environment. These methods go under the name *Reinforcement Learning* (RL) [69] and they are capable of learning solution policies through *interaction* with an environment or a simulation of it, which is often much easier to construct than specifying the model directly [69].

Q-Functions

Without a model of the transition distribution, it is not possible to extract an optimal policy π^* through Equation 3.4. RL methods often use a so called *Q-function* instead which is a mapping from both action and state to a value. The Q-value $Q^\pi(s,a)$ represents the expected value if the action a is executed in state s and then the policy π is followed.

Q-Learning

A simple variant of Reinforcement Learning is the *Q-Learning algorithm* which updates the Q-function after it has executed an action a in a state s and received the reward r in the following manner:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_N(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{3.5}$$

The α_N parameter is commonly set to:

$$\frac{1}{(1 + visits_N(s, a))} \tag{3.6}$$

where $visits_N(s, a)$ is the total number of times the action a has previously been executed in the state s.

The updated Q-function is guaranteed to converge to the correct value if, in the limit, all actions are executed an infinite number of times. This criteria can be satisfied by using suitable *exploration functions* that sometimes select (according to the current Q-function estimation) non-optimal actions. One such exploration function uses an ϵ-greedy policy which selects a random action with probability ϵ.

The Q-learning method described here is the most simple one possible since it uses a tabular representation of the Q-function. It is also very slow, especially when many sequential actions are required to receive a reward. If the model of the environment is known in advance, the Q-function can be represented and learned with function approximation techniques such as Neural Networks [9].

The full Q-learning algorithm is shown in Procedure 3.2.

Procedure 3.2 Q-LEARNING(γ)

```
Initialize $Q(s, a)$ arbitrarily, e.g. $Q(s, a) = 0$ for all $s$, $a$
repeat
  $s \leftarrow$ The current (initial) state
  repeat
    $a \leftarrow$ An action derived from $Q$ (e.g., $\epsilon$-gready)
    Execute $a$ and observe $r$ and $s'$
    $Q(s, a) \leftarrow Q(s, a) + \alpha_N(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  until $s$ is a terminal state
until Termination condition
Return $Q$
```

3.4.5 RL with Model Building

It is possible to combine the RL technique of learning a policy through interaction with the environment and the efficient methods for solving MDPs given a model. The trick is to learn a model of the environment simultaneously and use that to update the Q-function with. A straightforward method is to update all Q-values with a Q-function version of Value iteration after every interaction but that is a very computationally expensive method and more efficient methods exist.

Model Building

Since the environment is assumed to be fully observable, a Maximum Likelihood (ML) [2] estimation of the model can be applied. The transition function describes a multinomial probability distribution which can be ML estimated by keeping track of the number of times the transitions have occured.

Procedure 3.3 describes the details of how the model is learned given the executed current action a, previous state s', current state s and received reward r. $N_d(s, a, s')$ and $N_c(s, a)$ contain the counters that are needed to update the transition distribution $P(s'|s, a)$ and the expected reward $R(s, a, s')$ (which needs to include the resulting state s' in order to perform the Bellman update for Q-functions).

Procedure 3.3 UPDATEMODEL(a, s, s', r)

```
  $N_d(s,a,s') \leftarrow N_d(s,a,s') + 1$
  $N_c(s,a) \leftarrow N_c(s,a) + 1$
  $R_{sum}(s,a,s') \leftarrow R_{sum}(s,a,s') + r$
  $R(s,a,s') \leftarrow \frac{R_{sum}(s,a,s')}{N_c(s,a,s')}$
  $P(s'|s,a) \leftarrow \frac{N_d(s,a,s')}{N_c(s,a)}$
```

DynaQ Algorithm

The *DynaQ Algorithm* [68] is a simple but effective reinforcement learning algorithm that can be used together with UPDATEMODEL(). DynaQ uses both the update from Q-learning and the learned model to update the Q-function. The Q-function update with the model is performed with the Bellman update adapted for Q-functions:

$$Q(s,a) = \sum s' P(s'|s,a)(R(s',a,s) + \gamma \max_{a'} Q(s',a')) \tag{3.7}$$

which requires that $R(s', a, s)$ keeps track of the resulting state as well.

The Bellman update is performed N times with a randomly previously visited state s_r and action a_r. The DynaQ algorithm is shown in Procedure 3.4, which is a version of DynaQ that can return the Q-function when some termination condition is met. In Chapter 5, the DynaQ algorithm is used to perform planning and the termination condition is then that a certain number of simulation steps has been performed.

Procedure 3.4 DynaQ(γ)

```
Initialize $Q$
Set $N_d(s,a,s') = 0$, $N_c(s,a) = 0$, $R_{sum}(s,a,s') = 0$ for all $s$, $a$, $s'$
**repeat**
  $s \leftarrow$ Current state
  $a \leftarrow$ An action derived from $Q$ (e.g., $\epsilon$-gready)
  Execute action $a$. Observe resulting state $s'$ and the reward $r$
  $Q(s,a) \leftarrow Q(s,a) + \alpha_N(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
  UpdateModel($a$, $s$, $s'$, $r$)
  **for** $i \leftarrow 1$ to $N$ **do**
    $s_r \leftarrow$ A random, previouosly visited state
    $a_r \leftarrow$ A random, previouosly taken action in $s$
    $Q(s_r,a_r) \leftarrow \sum_{s'} P(s'|s_r,a_r)(R(s',a_r,s_r) + \gamma \max_{a'} Q(s',a'))$
  **end for**
**until** Termination condition
Return $Q$
```

Chapter 4

The DARE Method

An agent that operates in a dynamically and rapidly changing world must be able to continually augment its models at different levels of abstractions relative to the task at hand. In the introductory chapter it was mentioned that if an agent's problem models are allowed to vary during its operation depending on the current focus of attention (see Section 1.7) and the tradeoff between accuracy and feasibility, there are some important consequences that need to be taken care of. A generated or selected model can be too coarse for executing the corresponding solution with the available skills. Planning must therefore be performed on several dynamically created abstraction levels which means that the agent needs to continuously monitor the validity of the abstractions used. Replanning becomes an important instrument when abstractions become invalid or when unsuspected or unmodelled events occur. Due to the dynamic abstraction levels, a more flexible subproblem generation is needed than those currently present for fixed abstraction levels [47] [75] [6].

The purpose of this chapter is to describe a method [1] called DARE (**D**ynamic **A**bstraction-driven **R**eplanning and **E**xecution) which is designed to take the implications of dynamically generated problem models and abstraction levels into consideration. The DARE method is a template that needs to be filled with environment class-specific subproblem generation procedures, abstraction validity monitoring and problem solving techniques. For example, in Chapter 5, MDPs are generated and solved on several levels of abstraction with the help of task environment models and reinforcement learning. A primitive action in an abstract MDP's policy expands into a more detailed MDP which is how the implementation of DARE's subproblem generation procedure works for that environment class.

[1]It is called a *method* and not an *algorithm* because it is not detailed enough in order to analyse it from a viewpoint of running time etc. which is the convention used in [12]

4.1 Tasks and Beliefs

One of the main ideas of DARE is that the abstractions used by the agent are strongly influenced by its current tasks and beliefs:

- **Tasks:** A task describes an objective-reaching activity that denotes several possible sets of partially ordered skills. The denotation can either be direct or indirect through the use of other tasks. A task can often be performed in many different ways depending on the situation. The current tasks of an agent should strongly influence its abstractions. For example, if the agent is about to lift an object with its actuators, it should focus its attention on that object and other things that are relevant during such a task. The abstraction should filter out irrelevant details.

- **Beliefs:** An agent's beliefs are considered to be the set of models that the agent uses to represent its environment and itself with. Maps, logical knowledge bases and probability distributions are all examples of what are considered to be beliefs. An agent's current beliefs in combination with feedback from the environment should be used to determine what abstractions to use. For example, when more and more information becomes available about an agent's environment, the level of uncertainty decreases and the planning horizon can be increased and perhaps more simplified models can be used instead.

4.2 Overview of DARE

Figure 4.1 shows a rough sketch of how DARE works. The solid arrows specify the method's logic and the dotted ones specify information flow. The different parts of the method are summarized in the following list:

- **Find suitable problem models in current context:** The first step of DARE is to find a suitable model that can be used to perform or support planning. This step performs the dynamic abstraction with the help of the current beliefs and tasks. The current tasks of the agent are specified in an HSN structure which is described further in Section 4.5. The result of this step is a problem model (such as an MDP or a classical planning problem). Chapter 5 and 6 describe how this problem generation can be performed in both fully and partially observable task environments using optimization techniques. The information about the selected abstractions is stored in the HSN structure for later use.

- **Solve problem:** The problem model is then solved with a suitable solution technique and the solution is stored in the HSN structure. For example, if the problem model can and will be solved with a classical planner, the resulting sequential plan is stored in the HSN structure.

- **Refine solution?:** A decision has to be made whether the solution is concrete enough to be executed or if it must be refined first. It might be the case that a part of the solution is "travel to Stockholm" which might be refined into "go to car, enter car, drive towards intersection 1, turn left at intersection 1, ... ,exit car". Then a subset of this sequential plan can be refined further until it is decided that some action can be executed.

- **Specify subproblems:** If further refinement is necessary, subproblems must be specified. For example, if a sequential plan should be refined, the agent can choose to refine a subset of all actions. MDP policies can be refined in many ways such as refining all state-action pairs into a separate subproblem etc. When this decision is taken, the HSN structure is updated and DARE tries to find a suitable problem model (or models) for this particular refinement. This process is repeated until a concrete enough action turns up.

- **Models invalid or refinement needed on some level?:** Every abstraction must be closely monitored so that the assumptions are valid enough to trust the model. When an invalid abstraction is detected, DARE tries to find new problem models that are more accurate in the current situation.

- **Update skills:** The skills (see Section 2.5.3) that are currently executing must also be updated to reflect changes to the solutions stored in the HSN structure.

4.3 Execution Assumptions

Section 2.5 briefly described some of the available architectures for autonomous agents, primarily used for robotic applications. The collective term *skills* was used to describe the behavior-generating parts of the architecture which represents e.g. the Task Procedure Instances in MTA and Reactive Action Packages in RAPS.

In this chapter it is assumed that skills can execute in parallel with the DARE method and that it is possible to change the current executing skills dynamically, which is the case for the most common execution systems.

4.4 Refinement Assumptions

DARE heavily depends on the assumption that it is beneficial to partly *refine* a solution in order to construct new problem models that are more detailed or specialized to solve that particular part. This seems to be a very natural approach to problem solving and it has been used extensively

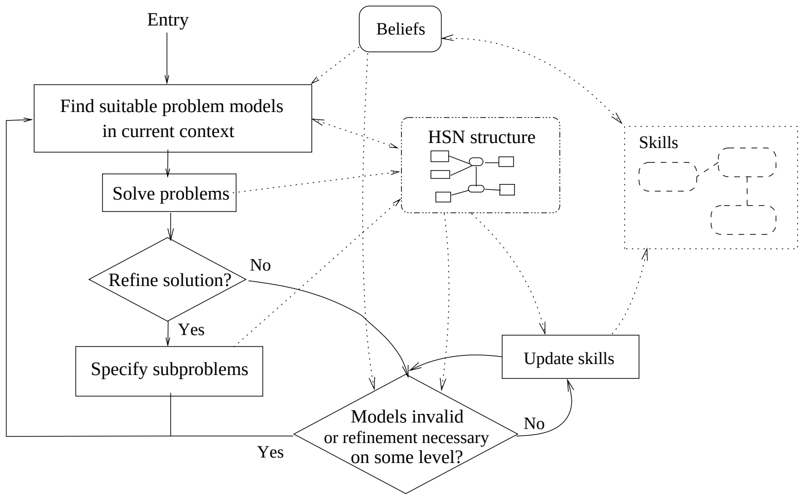

Figure 4.1: A rough sketch of how the (poll-based) DARE method works to control the currently (in parallel) executing skills in the system.

in HTN-planning systems like SIPE [75] and O-Plan [72] for many practial applications. It is also used within Hierarchical Reinforcement Learning [6] where extended versions of MDPs are used in hierarchies where a certain MDP can have primitive actions that actually execute a whole sub-MDP, which in a way generalizes HTN planning to stochastic environments.

All the existing systems that use task refinement use it in a very well-structured and well-understood way. Complexity results for HTN-planning have been investigated in [21] and it is now e.g. possible to compare the expressivity of HTN-planners with classical planners. Refinement has been viewed as an efficient method for humans to provide heuristics to a task environment. But for this to work, one has to know the exact workings of the environment, what predicates to use and how operators work. Refinement is almost always performed down to the most detailed level which is not possible in more open-ended and dynamic task environments.

Using dynamic abstraction while performing refinement makes things a lot more complicated. The task environment model is allowed to change and problems may be generated dynamically. Refinement in this context is not as crisply defined as with HTN-planners or hierarchical reinforcement learning methods. A simple example is when a classical planner is used to generate a solution to a dynamically generated problem. The steps in the solution might be refined in many different ways depending on the situation, making the solution process more flexible but also less defined in the general case due to the multitude of alternatives. One or several steps might be expanded into any suitable planning model (such as MDPs, POMDPs or even another classical planning problem) and it is therefore much more difficult to define the relationship between the abstraction levels than in currently existing task refinement systems.

Nevertheless, it is at least possible to measure the impact of a certain task refinement method if an agent with limited computational resources uses the method in an environment with a well-defined performance element that takes the deliberation time into account (as will be demonstrated in Chapter 5 and 6).

4.5 Hierarchical Solution Nodes

One of the implications of using dynamic abstraction for problem solving is the need to keep track of the abstractions and the resulting models to make sure that they are still useful. For example, if one of the assumptions is that an object in the environment is static, that assumption should become invalid when strong evidence to the contrary arrives. In the DARE method, the so called Hierarchical Solution Nodes (HSNs) keep track of the assumptions currently made and the conditions that can invalidate them.

Assumptions are typically made when problem models are constructed in a certain situation. Consider a UAS that is given the task of searching an area for certain objects such as fires, injured people, or certain vehicles. It must make assumptions about the environment in order to cope with the situation and task. The sensor input that the UAS can use to detect fires or bodies might for example come from a CCD camera, laser and an IR camera. It must be decided what the sensor input means and how it should be related to the UAS's current task. A possible abstraction is that bodies in the environment are represented with a position vector and the fires with some kind of area representation format e.g. polygons or special values in a grid.

The abstractions and the resulting problem models are used for the UAS's decision making and are therefore of utmost importance. It is important that the abstractions remain reasonably valid in order to make the problem models trustworthy. The abstractions must therefore be continuously checked or monitored and if some abstraction is considered invalid, the abstraction must be changed and the problem model that relies on it may have to change as well.

A HSN is used to store all this necessary information and can therefore be considered as a data structure that keeps track of abstractions, monitors, problem models and solutions.

A HSN is supposed to be used to generate or modify a set of skills that can be used to carry out solutions to a problem. Depending on the type of solution, level of detail in the abstractions and what types of skills that are available, it might also be necessary to store or generate information in the HSN that describes how sensor data should be obtained to guide the behavior.

As mentioned earlier, a problem specification that is generated through dynamic abstraction might be at a very high abstraction level and needs to be refined further before any available skill can execute some subset of the solution. A HSN can therefore contain a pointer to another HSN that represents a more refined solution to parts of the problem in its parent HSN. The set of HSNs and the parent-child relationships between them forms a *HSN structure* which represents the abstraction levels currently present in an autonomous agent.

Figure 4.2 illustrates a possible HSN structure where a set of abstractions and problem models are used at the same time with different assumptions about how the environment works.

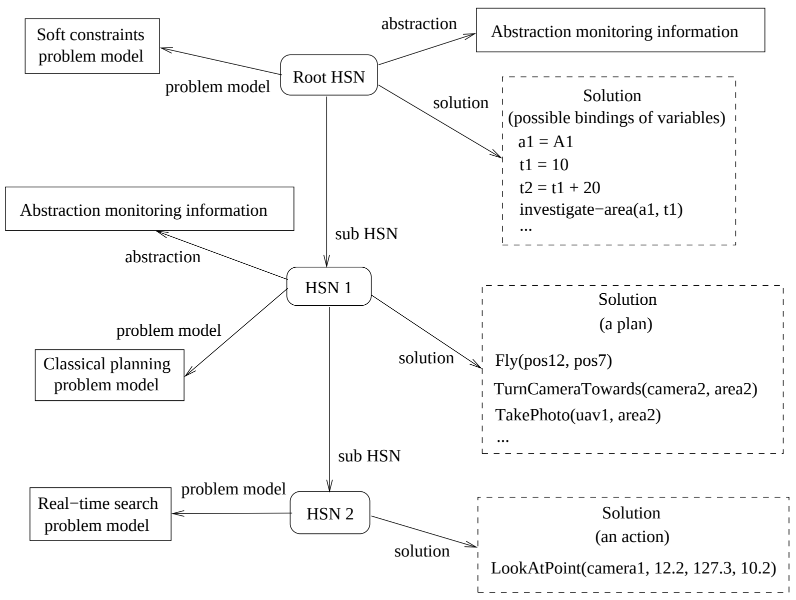
Soft constraints problem model
abstraction
Abstraction monitoring information
problem model
Root HSN
solution
Solution (possible bindings of variables)
a1 = A1
t1 = 10
t2 = t1 + 20
investigate−area(a1, t1)
...
Abstraction monitoring information
sub HSN
abstraction
HSN 1
Solution (a plan)
problem model
Classical planning problem model
solution
Fly(pos12, pos7)
TurnCameraTowards(camera2, area2)
TakePhoto(uav1, area2)
...
sub HSN
problem model
Real−time search problem model
HSN 2
Solution (an action)
solution
LookAtPoint(camera1, 12.2, 127.3, 10.2)

Figure 4.2: An HSN structure that combines different abstractions and problem models.

The soft constraints planning model, illustrated in Figure 4.2 as the root HSN's problem model, defines the overall mission which in this case is to monitor a certain area while at the same time try to fulfill all sorts of constraints that can typically be used in constraint-based planners such as HSTS [51] and ASPEN [28]. A solution to the planning model at this level is a set of possible variable bindings that determines what the agent should do next. Suppose that the solution is to investigate the areas A1, A3, and A2 in that order and then fly back to base to refuel.

Parts of that high-level solution are then refined according to the refinement assumption. Suppose that the action "investigate area A1" is the only action that is refined in this example. In this case, DARE constructs a classical planning model which takes the high level action into consideration but also adds additional details and other objectives that the higher abstraction did not consider.

In this thesis, it will be assumed that the model construction mechanism is capable of creating a reasonable model which can either guarantee the existence of a solution or fail and inform the "higher" abstraction level that the refinement was impossible to perform. This is not an issue for the case studies in Chapter 5 and 6 where the model type (MDPs and Depth-limited lookahead models) make it easy to always guarantee that a solution exists.

4.6 Subscription VS Poll

The DARE method can be implemented in different ways. Figure 4.1 describes a version where the abstractions are continuously monitored in the "Models invalid or refinement necessary on some level?" test. This means that the method is *poll-based*; the conditions are checked by the main thread in the method which makes it conceptually easy to understand and implement in simple environments.

It is also possible to create a *subscription-based* version of DARE where the conditions are checked by monitors that run in parallel with the main thread (or even in another process or on another computer). The main thread then sets up the monitors for the different abstractions that should be checked and the agent can then react to invalidated abstractions when they occur.

In the UAS Tech system, CORBA [59] is used as a communication middleware and the different software components are distributed on several computers. In such a distributed system, it might be more natural to implement the subscription-based DARE method instead due to the advantages of using event-based communication in such a context. It is therfore very likely that an implementation of DARE in the UAS Tech system will be subscription-based. However, it is much easier to present and understand single-threaded methods and the two partial implementations of DARE are both poll-based which supports repeatability of the experiments (see Chapters 5 and 6).

4.7 The Method

In this section, the poll-based version of the DARE method will be described in more detail.

4.7.1 Main

Procedure 4.1 shows the entry point of the poll-based DARE method which takes a set of beliefs (see Section 4.1) *Bel* as input. *Bel* represents all available information that is currently accessible to the agent. This set is allowed to change dynamically depending on e.g. changing conditions and different focus of attention. A skill might for example need specific knowledge that is calculated elsewhere.

Procedure 4.1 DARE(Bel)

```
1: rootHSN ← new Hierarchical Solution Node
2: DYNABSSOLVE(rootHSN, Bel)
3: while not FINISHED(Bel) do
4:     REPLANIFNECCESSARY(rootHSN, Bel)
5: end while
```

DARE initially constructs the *Root HSN*, which represents the highest abstraction and decision level in the system. No skills are assumed to be executing in the system at this moment. It then calls the DYNABSSOLVE() procedure (see Section 4.7.2) to generate the first HSN structure that depends on the current beliefs *Bel* and starts up the skills that will execute the initial solutions to the problems. Figure 4.2 illustrates an example of what a resulting HSN structure might look like after calling DYNABSSOLVE().

The next task is to actively monitor the abstractions and problem models to see whether they need to be changed. This task is performed in a loop [2] that is executed until the agent considers that it is finished. Whether the loop finishes or not by the agent's initiative depends on the application. Some tasks naturally have a well-defined end such as when a UAS is sent out to take a set of pictures of a building structure and return to base, land autonomously and turn off the engine. Other tasks are more continuous in nature such as flying a patrol route while refueling when necessary. In that case, a human operator might trigger the finished condition.

Inside the loop, DARE calls the REPLANIFNECCESSARY() procedure (see Section 4.7.4) which continuously checks that the current abstractions are valid and performs replanning (by calling DYNABSSOLVE()) if this is necessary.

4.7.2 DynabsSolve

The DYNABSSOLVE() procedure (listed in Procedure 4.2) can be considered the central part of DARE since it performs (or coordinates) the following important tasks:

[2] In the subscription-based version of the method, the loop is simply replaced with a wait for input signals from the monitors.

Procedure 4.2 DYNABSSOLVE(hsn, Bel)

```
1: abstraction(hsn) ← FINDABSTRACTION(Bel)
2: problem(hsn) ← GENERATEPROBLEM(abstraction(hsn), Bel)
3: solution(hsn) ← SOLVE(problem(hsn), Bel)
4: MODIFYSKILLS(hsn, Bel)
5: CREATESUBPROBLEMS(hsn, Bel)
```

- **Selection of a suitable abstraction**: An abstraction is selected or generated by a call to FINDABSTRACTION() which e.g. answers questions such as: Should the vehicle v be viewed as a stationary 3-dimensional object or perhaps as a point with position and velocity? Should the environment be considered stochastic or deterministic? The best decision should be the one that gives the best performance of the task given the available algorithms and computational resources.

- **Generation of a problem model**: The abstraction is then used to generate a problem model which should depend on the chosen abstractions. DARE performs this step with a call to GENERATEPROBLEM(). If e.g. the abstractions determine that the environment should be considered deterministic, it might be possible to generate a classical planning model etc.

- **Solving the problem**: The SOLVE() procedure is then used to solve the generated problem model with a suitable solution algorithm.

- **Modifying the skills**: The skills executing in the system may have to be modified according to the solution. This part is performed with a call to MODIFYSKILLS().

- **Creating subproblems**: If it is considered necessary to refine the solution, CREATESUBPROBLEMS() creates subproblems and adds a sub HSN to the current HSN (see Section 4.7.3).

In Chapters 5 and 6, concrete implementations of DYNABSSOLVE() will be presented both where the generated problem is an MDP and a partially observable DDN.

4.7.3 CreateSubProblems

The main assumption within DARE is that a solution at one level can be refined into subproblems that can either be solved with a solution method or executed directly with some parameterized skill. It was argued in Section 4.4 that this is a good thing to do.

The task refinement in DARE is performed with the CREATESUBPROBLEMS() procedure (see Procedure 4.3. In that procedure, there is a call to

CUTOFFTEST() which checks whether any more refinement of the current solution should be performed. This can depend on many things but should ultimately be tuned by the expected performance that it yields if used.

Procedure 4.3 CREATESUBPROBLEMS(hsn, Bel)

```
if CUTOFFTEST(hsn, Bel) then
    subNode(hsn) ← nil
else
    subNode(hsn) ← CREATESUBHSN(hsn, Bel)
    DYNABSSOLVE(subNode(hsn))
end if
```

A quite general cutoff criteria might be that there exists skills that reliably can execute the solution at the current abstraction level and more refinements would not yield a better performance (due to the extra computational cost to call DYNABSSOLVE() once more).

4.7.4 ReplanIfNecessary

The poll-based DARE method calls REPLANIFNECESSARY() continuously to check that the current abstractions are reasonable and to update the HSN structure and modify the currently running skills if necessary.

Procedure 4.4 REPLANIFNECESSARY(hsn, Bel)

```
if hsn == nil then
    return {No replanning necessary}
end if
if ABSTRACTIONINVALID(hsn, Bel) then
    subNode(hsn) ← nil
    DYNABSSOLVE(hsn, Bel) {Replan}
else
    if KEEPSUBHSN(hsn, Bel) then
        REPLANIFNECESSARY(subNode(hsn), Bel)
    else
        CREATESUBPROBLEMS(hsn, Bel)
    end if
end if
```

The HSN structure is then traversed from the root and down and the abstractions are checked for validity and other criteria such as if a partial solution should be extended. The abstraction validity check is performed with a call to ABSTRACTIONINVALID() and DYNABSSOLVE() is called if a new abstraction is considered necessary.

Examples of reasons for changing abstractions can be that something should be viewed in a different way depending on new information. An object can for example become totally irrelevant for performing a certain task. Examples of this are given in Chapter 5 and 6 where an observation target is considered irrelevent after it has been classified and will not be considered when the problem models are constructed. Abstractions may also have to change when an action that was previously considered deterministic turns out to be unreliable for some reason (perhaps a mechanical error). The available processing power for the decision process may also change for different reasons and then other types of abstractions are more suitable. Other practical decisions such as sampling or generation rate of data may also have to be modified dynamically depending on the available computational resources and the requirements of the current problem models.

Even if the abstractions are valid in the current HSN, it may still be the case that the sub HSN must be replaced or modified which is checked with a call to keepSubHSN(). An example of such a situation is when only parts of the solution have been refined in the sub HSN and it is necessary to refine some more, possibly due to the agent's execution.

If the sub HSN is kept, replanIfNecessary() is called recursively and applied to that HSN.

4.8 Discussion

This chapter described the poll-based DARE method which is a template for performing planning and execution with dynamic abstractions.

The method is very abstract and many things have to be instantiated before it can be used. In this thesis, two instantiations have been constructed which demonstrate all parts of the method.

DARE is very useful in dynamic task environments where it is not possible to represent the different parts of the world completely at all times both during execution and planning and when it is beneficial to change representation when necessary to adapt to changing conditions. However, for well-specified task environment models which do not include any fundamental surprises, it is always possible to construct better specialized agents that can outperform DARE. The strengths of DARE lies in its potential of handling surprises that are not part of any fixed task environment model. A task environment that is part of the real world can make any agent with a fixed model fail miserably because of its inability to reason about detailed parts of its environment. An agent driven by a sophisticated DARE implementation should be able to test different ways of viewing its environment and thereby become much more robust. This would probably require a large effort where a formal language for describing abstractions and model constructions would help, which is a topic for future work (see Section 7.1).

The presented version of DARE can be further improved. Generated solutions are currently thrown away if an abstraction is considered invalid which can be very wasteful. It should be possible to reuse previously generated solutions with some kind of *case based reasoning* [1] or other machine learning methods.

Chapter 5

Case Study I

The poll-based DARE method, that was presented in the previous chapter, is very abstract and a lot of environment class-dependent work is needed to implement it. The reason for this abstract presentation is that the core principles of DARE can be applied to a large class of task environments and problem models that benefit from dynamic abstraction to focus the attention on the most important parts during the decision making. In this chapter, a concrete implementation of the method is presented that has been adapted to a continuous, stochastic and fully observable environment class inspired by the UAS Tech system. The environment class contains a utility-based agent that receives rewards when it classifies dynamic observation targets or reaches a certain target area. At the same time it tries to avoid dangers that inflict negative rewards if the agent is too close to them.

In this environment class, MDPs (see Section 3.4) are used as the planning model at every abstraction level and solved with reinforcement learning together with task environment models that are possible to simulate. The task environment models are implemented with fully observable DDNs (see Chapter 3).

Parts of the results in this chapter have been published in [57].

5.1 Task Environment Class

The instances of the task environment class contains a single agent that operates in a sequential, stochastic, continuous and fully observable 2D environment (see Figure 5.1) which can contain any number of the following elements:

- *Finish areas* which are rectangular areas where the agent can safely finish its current task or subtask. Each area is associated with a reward that can be used to specify several competing target locations.

- *Road networks* which are undirected graphs where the edges are line segments that can be traversed by different types of external agents (see next item).
- *External agents* which are objects that can either move freely (assuming that it is a point with a certain mass and maximum acceleration) in the environment or bound to a road network. The external agents can either be *dangers* or *observation targets*. Dangers should be avoided by the agent and they are associated with a certain negative reward that the agent receives when it comes too close to it. The agent can try to classify an observation target, if it is close enough, and if it is successfull it receives a positive reward.

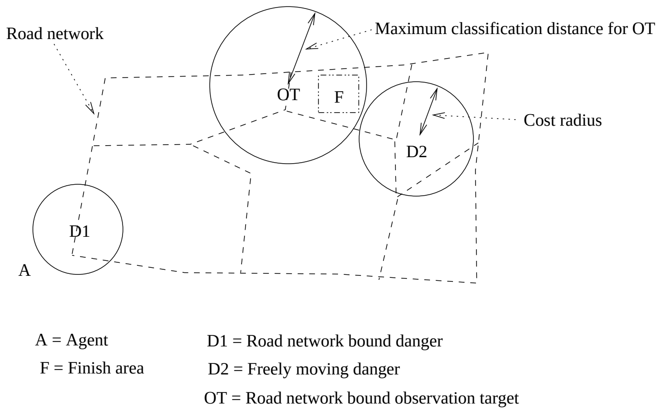
Road network
Maximum classification distance for OT
OT
F
Cost radius
D2
D1
A
A = Agent
F = Finish area
D1 = Road network bound danger
D2 = Freely moving danger
OT = Road network bound observation target

Figure 5.1: An instance of the fully observable UAS environment class. The cost radius is described in Section 5.2.1

The UAS agent can perform one of the following actions:

- *Move* in eight possible directions (also called "kings moves") with a certain speed (10 m/s in the implementation).
- *Wait* at the current position.
- *Finish at area* which means that the UAS agent moves towards a finish area and finishes when it is reached. There is one such action for every existing finish area.
- *Try to classify* an observation target by moving towards it and continuously perform the classification. This action models a more detailed sensor action which the agent can use to extract more information about a target than its position.

This task environment class will be called the *fully observable UAS environment class.*

5.2 Task Environment Model

The task environment models are implemented by fully observable DDNs whose structure is shown in Figure 5.2. The external agents are assumed to be independent of each other. A DDN can be used for simulation with the help of a random number generator where the probability distributions are sampled. Such a task environment model for simulation and evaluation of the agent is supposed to have a fixed time step length dt. When a DDN is used by the agent during the planning phase, dt is determined dynamically (see Section 5.4.2). An agent receives a reward of -1 for each action it executes during one second which means e.g. that moving 100 meters gives a total reward of -10 if no dangers are around (see Section 5.2.1).

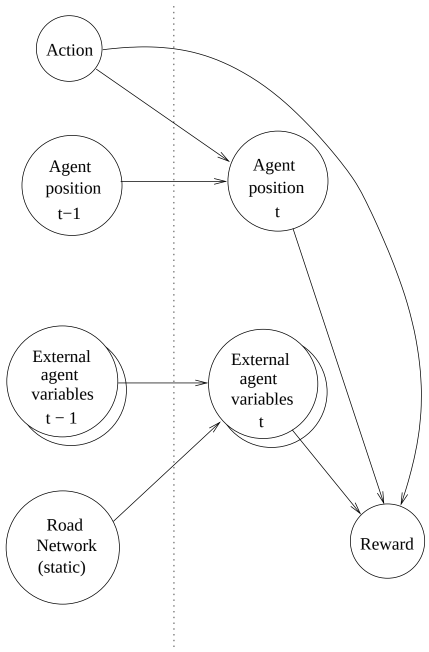

Figure 5.2: The general structure of the fully observable DDN that implements the task environment model.

Figure 5.3 shows the relationship between the state variables in a freely moving external agent. The speed and direction is assumed to change randomly according to the Gaussian distributions $N(0, \sigma_{dt,vel})$ and $N(0, \sigma_{dt,dir})$ where $\sigma_{dt,*}$ $(* \in \{dir, vel\})$ depends on the time step length used. It is assumed that the standard deviation is equal to $\sigma_{1,*}$ for a dt equal to one second. In order to make the same standard deviation for N steps of length $1/N$ seconds, $\sigma_{dt,*}$ is set to $\sigma_{1,*}\sqrt{dt}$.

The road network bound agents (see Figure 5.4) are modelled implicitly

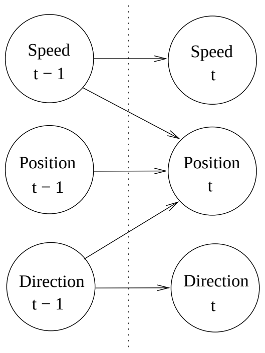

Figure 5.3: The DBN for a freely moving external agent.

by first generating a random length increase depending on the current velocity. The road network is then used to determine where the agent goes by using a uniform distribution at junctions. The distribution for the current road segment and segment length is implicitly determined (see Section 2.2.4) by a program that follows the current road segment to a junction and then samples the next way to go until the same distance has been covered as in the “distance to go” variable. This is possible since the solution method used (reinforcement learning) does not need an explicit distribution or density function.

5.2.1 Danger Rewards

Ideally, the (negative) reward received from a danger *do* during the time $t_{\text{now}} - dt$ to t_{now} can be calculated as follows:

$$R_{do} = \int_{t_{\text{now}}-dt}^{t_{\text{now}}} min(-C_{max} + \frac{C_{max}}{C_R}|\mathbf{p_a}(\tau) - \mathbf{p_{do}}(\tau)|, 0)d\tau \qquad (5.1)$$

where $\mathbf{p_a}(\tau)$ and $\mathbf{p_{do}}(\tau)$ are the functions that describe the movement of the agent and the danger. C_R is the *cost radius* which determines the distance from the danger where the reward is zero. C_{max} is the highest negative reward that can be received per second. dt is quite small and an approximation of R_{do} is used in the implementation by the following formula:

$$R_{do} = dt \cdot min(-C_{max} + \frac{C_{max}}{C_R}d_{min}, 0) \qquad (5.2)$$

Here, d_{min} is the minimum distance between the danger and the agent during $[t_{\text{now}} - dt, t_{\text{now}}]$ which makes the approximation a pessimistic one (from the agent's point of view).

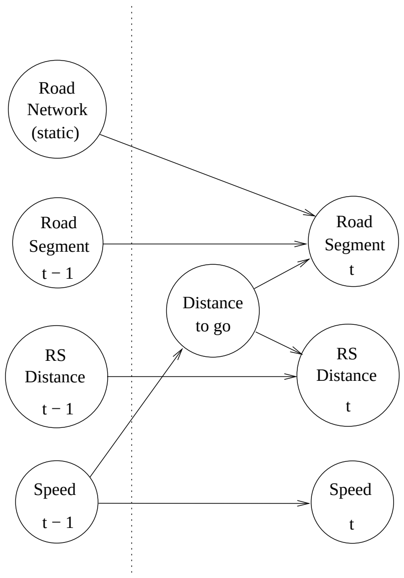

Figure 5.4: The DBN for a road network bound observation target.

5.2.2 Observation Target Rewards

For every observation target in the task environment, the DDN includes a boolean variable cl_{ot} which specifies if ot has been classified. The probability $P(cl_{ot,1}|d)$ of classifying ot from a distance d during one second is specified with a so called *Continuous-time Markov Process* [4] with only two possible states and with the intensity λ_d of going from "not classified" to "classified". The intensity decreases linearly from a maximum value $\lambda_{ot,max}$ to zero at the maximum classification distance $d_{ot,max}$ and beyond. Every instantiation of a DDN with a time step of dt uses a probability distribution $P(cl_{ot,dt}|d)$ (see Equation 5.3) which determines the probability of classifying ot if the classification action is performed for a duration of dt.

$$P(cl_{ot} = true|d, dt) = 1 - e^{-\lambda_d dt} \tag{5.3}$$

If the classification succeeds, the agent receives a reward $R_{cl,ot}$ that is independent of the distance to the observation target.

5.3 Skills

There are four different parameterized skills available which can be used to execute the actions described in Section 5.1. Only one skill at a time can be executed, which makes it very easy to implement the MODIFYSKILLS()

procedure which is called from DYNABSSOLVE() in DARE (see Section 4.7.2). In DYNABSSOLVE(), the MODIFYSKILLS() procedure is executed every time a solution has been found, making it possible to structure and coordinate skills that operate with different abstraction levels. This is not necessary for the implementation described in this chapter where there can only be exactly one skill executing at all time, which is the one that corresponds to the solution at the "lowest" abstraction level.

The basic movement skills simply make the agent move in one of eight possible directions (kings move directions) until the skill is terminated by MODIFYSKILLS(). The *Finish at area* skill moves the agent towards the closest point in a finish area and finishes the execution when the agent is within that area. The *Try classify* skill moves the agent towards an observation target and tries to classify it at the same time. The probability of success is specified in Section 5.2.2.

5.4 DARE Implementation

This section will describe in detail how the different parts of the DARE method are implemented for the fully observable UAS environment class. There are several questions that need to be answered when DARE is implemented such as what type of problem models to use and how they are generated and solved.

5.4.1 Problem Models

The task environment model presented in Section 5.2 has continuous state variables which means that it is difficult to use directly for planning. Two common methods will be considered in this thesis to perform planning in such environment classes: Depth-limited lookahead and Reinforcement Learning (RL). Depth-limited lookahead will be explored in Chapter 6 and RL is used in this case because of the fully observable state variables and the opportunity to demonstrate a simple but fully working dynamic abstraction method (see Section 5.4.2).

As mentioned in Section 3.4.4, RL methods assume that the environment can be represented with an MDP but it is not neccessary to provide detailed transition and reward distributions in advance. In the fully observable UAS environment class, a task environment model in the form of a DDN is available to the agent but this has a continuous state space which can not directly be used without either some function approximation of the Q-function and/or discretization of the state space. Function approximation is avoided in this case to make it possible to study the dynamic abstraction method in isolation.

5.4.2 Dynamic Abstraction

The main idea of dynamic abstraction is to dynamically generate models in a way that suits the current circumstances in the best possible way (see Section 1.8). The abstractions chosen should also depend on the available computational power.

In this implementation of DARE, much of the abstraction is already decided. MDPs are used to represent the planning models which means that the environment is considered to be stochastic but fully observable. In a more flexible and capable dynamic abstraction "module", this type of reasoning should be performed automatically. This is currently considered as future work and is further discussed in Section 7.1. The dynamic abstraction in this implementation will be concerned with how the state space S should look like when the MDP is solved and how the mapping from the task environment model's state variables to S is done.

The most common method when creating a state space is to use a fixed discretization. This implementation will however change the discretization depending on what parts of the environment are considered most relevant at the moment. A danger that is very close to the agent should for example be considered more relevant than a danger that is very far away and the individual possible negative rewards that they can inflict should be taken into consideration.

The main idea of the dynamic construction of S is then to focus more on the more relevant objects and state variables in the environment class by giving them a greater number of possible discrete values. At the same time, $|S|$ is limited by a constant, giving the less relevant objects and state variables fewer possible discrete values.

Relevance

The number of possible discrete values for an external agent is determined by defining an optimization problem over possible discretizations. The utility of a discretization is specified to depend on the so called *relevance* of the different external agents. The relevance of an external agent decreases with distance d and also depends on the cost radius $C_{max,do}$ for dangers and maximum classification distance $R_{cl,ot}$ for observation targets.

The relevance function for dangers *do* is defined as follows:

$$Rel_{do}(d) = C_{max,do}e^{-\alpha_{do} \cdot d} \tag{5.4}$$

where α_{do} is set to a value that makes Rel_{do} equal to 10 percent of its maximum value at the cost radius. By using the exponential function, dangers will never be totally irrelevant. Other functions are of course possible but this seems to work well in the fully observable UAS environment class.

The relevance function Rel_{ot} for observation targets is similar to Rel_{do} except that $R_{cl,ot}$ is used instead of $C_{max,do}$ and an extra factor γ_{ot} (see

Section 5.5.1) is multiplied with Rel_{ot} which makes it possible to adjust the relevance when the continuous negative reward received from dangers is compared with the "one shot" reward received when an observation target is classified.

The relevance function for observation targets ot is then defined as:

$$Rel_{ot}(d) = \gamma_{ot} R_{cl,ot} e^{-\alpha_{ot} \cdot d} \tag{5.5}$$

where α_{ot} is (similarly to α_{do}) set to a value that makes Rel_{ot} take the value of $0.1 \cdot R_{cl,ot}$ at the maximum observation distance.

The state variables of the agent itself must also be represented in S. It is simply assumed that the relevance Rel_{XY} for the agent's position variable (X and Y coordinate) is the same as the sum of the external agents' relevances. This means roughly that the agent should represent its own state variables in S as much as its environment's.

Discretization Optimization

The relevance functions are then used to define the utility of a discretization U_{disc} using the following formula:

$$U_{disc} = \sum_{i \in OT \cup DO \cup \{X,Y\}} (1 + \frac{Rel_i}{|S_i|})^{-1} \tag{5.6}$$

where S_i is the number of discrete values that are assigned to the object or state variable i, OT is the set of observation targets, DO is the set of dangers and $\{X, Y\}$ is the agent's position variables. This particular formula was chosen because it seems to distribute the number of discrete values in a reasonable way in the sense that the increase of U_{disc} decrease for higher number of values. Other possible utility functions include variants and combinations of the sigmoid and the tangent function.

A maximum state space size S_{max} is used to limit the size of S, because it is then possible to partly control the time that is necessary to provide a reasonably good policy. The total state space size is calculated by multiplying all the state contributions S_i.

A reasonable state distribution for a discretization is found in the current implementation by performing Hillclimbing search (see Section 2.4), maximizing the utility distribution U_{disc} from the initial state where all objects and features have only one state each.

Clustering

When the optimization is done, the discretization must also define the mapping from state variables to the different states. The agent's position variables are mapped to a standard grid with a width and height determined by the state distribution. The mapping for the external agent's state variables are determined dynamically with k-means clustering [33]. The k-means clustering algorithm is shown in Procedure 5.1 where K is set to the value received from the state distribution and I is the set of instances.

Procedure 5.1 KMEANCLUSTER(I, K)

```
1: $Centroids \leftarrow$ K number of random instances from $I$
2: repeat
3:   for all $i \in I$ do
4:     Calculate each $i$'s closest centroid $i_c$
5:   end for
6:   for all $c \in Centroids$ do
7:     Calculate the center of $c$ given its assigned instances $I_c$
8:     Assign a new centroid $c_{new}$ that is closest to the center of $c$
9:   end for
10: until All centroids stay the same
11: Return $Centroids$
```

The set of instances I are generated by sampling the task environment model from the current state N_s runs with N_p samples in each run. The so called *temporal horizon* $T_{horiz,ddn}$ of the DDN defines how far ahead in time the task environment simulation and instance generation will be performed. The temporal horizon for a DDN with width w and height h is $v_A^{-1} max(w, h)$ where v_A is the agent's speed. The number of runs N_s is always set to 10 and the number of samples at each run N_p is always 100 which gives a total of 1000 instances for each external agent.

The temporal horizon together with the standard grid determines the dt parameter that is used to construct a fixed time step DDN. The time step dt is set to $\gamma_{dt} min(w_g, h_g)$ where w_g and h_g is the width and height of a standard grid cell and γ_{dt} is a constant factor that determines how long dt should be relative to the grid cell size. γ_{dt} was set to 0.1 during the experiments (see Section 5.5).

Figure 5.5 illustrates a typical result after the discretization optimization and clustering.

5.4.3 Solution Method

The discretization, determined by the dynamic abstraction method described in Section 5.4.2, can then be used for planning. Since the transition and reward distributions for the given discretization is unknown and difficult to calculate exactly from the DDN, the DynaQ (see Section 3.4.5) reinforcement learning method is used to solve the induced MDP. DynaQ is therefore used for both implementing the GENERATEPROBLEM() (transition and reward distribution) and the SOLVE() procedures in DARE.

In the implementation, the ϵ-greedy exploration function (see Section 3.4.4) is used with ϵ set to 0.1 and the number of planning steps performed

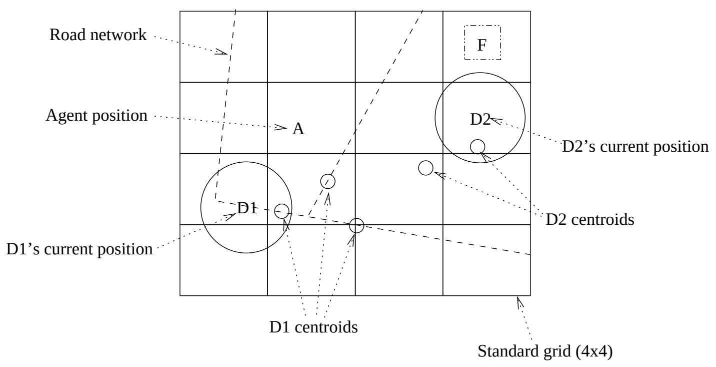

Figure 5.5: An example discretization after discretization optimization and clustering. The total state space is $16 \cdot 3 \cdot 2 + 1 = 97$ where the standard grid contributes with 16, D1 with 3, D2 with 2 and the finish area with 1 possible discrete values. D1 is a road network bound danger that is moving to the right and D2 is a freely moving danger that moves towards south-west.

in DynaQ is set to 5. The step length time is determined by the width of the cells in the standard grid with respect to the agents speed.

DynaQ was originally designed to be used for continuous interaction with an environment and not to get a solution within a certain time. It can easily be turned into an *anytime* algorithm [15] by letting it run for a certain amount of time or number of execution steps. The question is then: How long should it run before the Q-function can be used for execution? The question is central and important for the use of dynamic abstraction for problem solving because the time used for problem solving is important when the tradeoff between feasibility and accuracy of the planning model is determined. Section 5.5 presents some experiments where this tradeoff is specified when the dynamics of the environment, S, execution speed and number of simulation steps that DynaQ performs are taken into account.

Procedure 5.2 shows the implementation of DYNABSSOLVE() in the fully observable UAS environment class. FINDDISCDIST() implements the discretization optimization described in Section 5.4.2. GENERATEINSTANCES() performs the collection of instances to the KMEANCLUSTER() algorithm that returns the set of centroids for all external agents. DYNAQSTEPLIMITED() is an implementation of DynaQ where the number of simulation steps is limited which determines the termination condition in DynaQ (Procedure 3.4 on page 36). One of the experiments in Section 5.5 determines the optimal balance between state space size and number of simulation steps for the implementation.

The dynamic abstraction and solution method has now been defined when a certain task environment model is given to the agent. For task environment models with many external agents the discretization becomes

Procedure 5.2 DYNABSSOLVE(hsn, V)

```
1: discDist(hsn) $\leftarrow$ FINDDISCDIST(V)
2: for all External agents $ea$ do
3:     $I \leftarrow$ GENERATEINSTANCES(TEModel(hsn), V)
4:     clusters($ea$, hsn) $\leftarrow$ KMEANCLUSTER(I, discDist($ea$, hsn))
5: end for
6: solution(hsn) $\leftarrow$ DYNAQSTEPLIMITED(hsn)
7: timeStamp(hsn) $\leftarrow$ CURRENTTIME()
8: CREATESUBPROBLEMS(hsn)
9: if subNode(hsn) = nil then
10:     Set the current skill according to solution(hsn)
11: end if
```

very coarse and the solution steps can take a long time to execute. The refinement assumption (see Section 4.4) states that it might be beneficial to refine such coarse solution steps into subproblems with the CREATESUBPROBLEMS() Procedure in DARE. This procedure is described in the next section.

5.4.4 Subproblem Generation

The implementation of DARE's CREATESUBPROBLEMS() for the fully observable UAS environment class creates subproblems by generating new DDNs that are determined by taking the solution policy into consideration. If the solution e.g. has a *Move East* action specified for the current state, the sub DDN for that subproblem is created with an added finish area to the east of the agent. Figure 5.6 illustrates the different types of submodels that can be constructed. The idea is to use that submodel to generate a more detailed solution for moving the agent to the east, ignoring the other parts of the environment at the moment. The relation between the solution on one abstraction level and the refined solution is then specified with dynamically generated task environment models.

With this implementation of CREATESUBPROBLEMS() it is possible, in theory, to refine solutions indefinitely which is not acceptable. The CUTOFFTEST() in DARE is in this case used to stop the refinement when the solution is considered detailed enough. In this fully observable UAS environment class, the cutoff is made when the sub DDN's width or height is smaller than a certain threshold (50 meters in this case) or when the DDN does not contain any external agents.

There is also a question of *how much* of the solution to refine. At one extreme, every state/action pair in the policy can be refined, leading to $|S||A|$ number of refinements. A more likely situation is that only a strict subset of the solution is refined due to demands of a reasonable response time. The current implementation only refines the current state and the

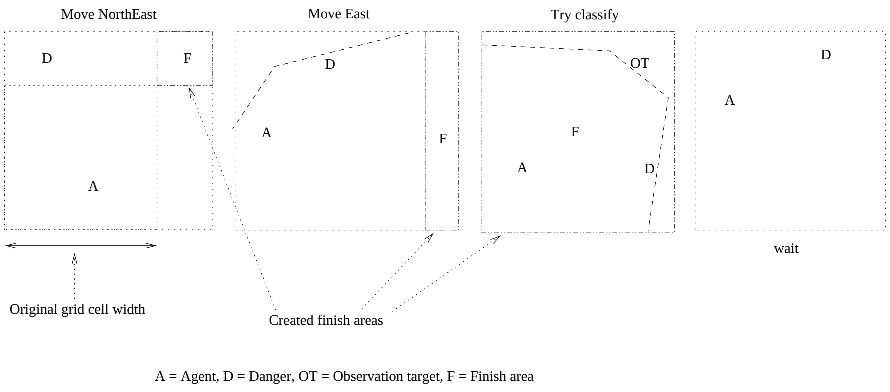

Figure 5.6: Examples of sub DDNs that can be created for some of the agent's actions. The new DDN's width is determined by the size of the standard grid cells and the type of action that the DDN represents. New finish areas are constructed in the generated DDN that represents the sub-problem's goal. Notice that the agent is allowed to finish before it has classified the observation target, which means that the *Try Classify* sub-problem models the possibility of "giving up" if it is considered to be too dangerous (costly). The sub DDNs for *Move NorthWest*, *Move West* etc. are created in similar ways.

solution policy's action in that state.

The implementation of CREATESUBPROBLEMS() for the fully observable UAS environment class is shown in Procedure 5.3 where CREATESUBHSN() constructs a DDN according to the current solution policy.

Procedure 5.3 CREATESUBPROBLEMS(hsn, V)

```
1: if not CUTOFFTEST(hsn, V) then
2:     subNode(hsn) ← CREATESUBHSN(hsn, V)
3:     DYNABSSOLVE(subNode(hsn))
4: end if
```

5.4.5 Replanning Conditions

After DYNABSSOLVE() has generated an initial solution in the poll-based DARE method, a loop is entered (see Procedure 4.1 on page 44) where the REPLANIFNECESSARY() is called to continuously check if any replanning needs to be performed due to unsuitable abstractions or other conditions.

The call to ABSTRACTIONINVALID() is made to check if the currently used abstractions are invalid and need to be replaced with a call to DYNABSSOLVE(). The implementation of ABSTRACTIONINVALID() for the fully

Procedure 5.4 cutoffTest(hsn, V)

```
1: if No of external agents in the (not yet created) TEModel(subNode(hsn)) = 0 then
2:     Return true
3: else if The width or heigth of TEModel(subNode(hsn)) < 50 meters then
4:     Return true
5: else
6:     Return false
7: end if
```

Procedure 5.5 createSubHSN(hsn, V)

```
1: Create a new HSN newHSN
2: TEModel(newHSN) ← The task environment model that corresponds to the action in solution(hsn)
3: Return newHSN
```

observable UAS environment class is shown in Procedure 5.7 which performs the optimization of the state distribution and checks if it differs too much from the current one. The constant DD_{max} determines how much the normalized state distributions can differ before the abstraction is considered invalid. The HSN structure also keeps track of the time when the abstractions first was used. An abstraction is also considered invalid if it has been used more than a fraction α_g of the temporal horizon for the task.

If the abstraction is considered OK, the keepSubHSN() procedure is called to check if the current sub HSN should be kept or not. In the implementation, if the agent executes a solution to a subproblem that leads to a goal or subgoal, a new subproblem must be generated and solved with a call to createSubProblems(). A new subproblem can also be generated if a state change occurs and the solution policy specifies that a different action should be executed than the one used to generate the subproblem. If for example an external agent makes the state change and the best action is considered to be *Move South* instead of *Move East*, a new subproblem is created that corresponds to the *Move South* action and dynabsSolve() is called with the corresponding sub DDN.

5.5 Experiments

A set of experiments have been performed with the implementation to test the viability of this type of dynamic abstraction method when used in the fully observable UAS environment class.

Procedure 5.6 REPLANIFNECESSARY(hsn, V)

```
if hsn == nil then
  return {No replanning necessary}
end if
if ABSTRACTIONINVALID(hsn, V) then
  set subNode(hsn) to nil
  DYNABSSOLVE(hsn, V) {Replan}
else
  if KEEPSUBHSN(hsn, V) then
    REPLANIFNECESSARY(subNode(hsn), V)
  else
    CREATESUBPROBLEMS(hsn, V)
  end if
end if
```

Procedure 5.7 ABSTRACTIONINVALID(hsn, V)

```
if (CURRENTTIME() - timestamp(hsn)) $> T_{replan}$ then
  Return true
end if
diff $\leftarrow$ | stateDist(hsn) - FINDDISCDIST(hsn) |
if diff /|diff| $> DD_{max}$ then
  Return true
else
  Return false
end if
```

5.5.1 Setup

All experiments were performed in a so called *simulated dynamic mode* which means that the environment evolved during the agent's deliberation, which is important to simulate in general if the task environment (see Section 1.2) is actually dynamic. The deliberation time was determined by the number of steps the DynaQ algorithm performs when the planning is performed at the abstraction levels, and is therefore assumed to be a function of the number of steps.

A set of 500 randomly generated test task environments were generated and used in the experiments where the same random seed was used every time in a given environment. This means that the external agents behaved in the same way every time given a certain environment number, which reduced the variance in comparison tests [36]. The agent had access to all the parameters of the task environment except for the actual outcomes of the random number generators. The step lengths during evaluation were also different from the models that the agent used during planning.

Each task environment had 1-5 observation targets with a random observation reward between 10 and 50, 1-5 dangers with a C_{max} (maximum negative reward) of 10 per second, and 1-3 finish areas with a random reward between 10 and 30. The starting position of the agent were randomized as well in an area which is always 400x300 meters.

State Space vs Simulation Steps

The maximum state space size S_{max} during discretization optimization is supposed to approximately determine the time it takes to solve the generated MDP. A larger S_{max} makes the solution more detailed and probably gives a better performance, but only if DynaQStepLimited() is allowed to take a sufficiently large number of steps. More steps take more time which makes the total performace go down due to the simulated dynamics model which could yield an optimal configuration of those two parameters. Figure 5.7 shows the result when the number of simulation steps and S_{max} are varied. It seems like there is a quite large range of S_{max} and number of simulation steps that yields acceptable performance, as long as neither of them are set too low. This is a good sign that indicates that detailed parameter tuning is not vital for the performance. It is also important to point out that S_{max} specifies the *maximum* possible number of states that can be discovered during the planning. In practice, the number of discovered states is sometimes much less, especially when S_{max} is very large (see Table 5.1).

Relevance Factor for Observation Targets

Section 5.4.2 introduced the relevance factor γ_{ot} for observation targets that is used to determine the relevance when so called "one-shot" rewards received from classifying observation targets are compared to the continuous

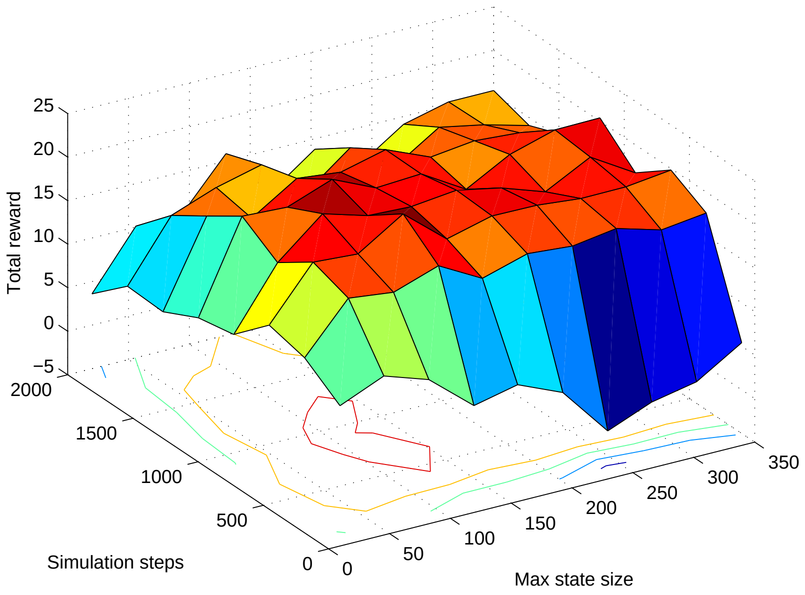

Figure 5.7: The result when the number of simulation steps and S_{max} are varied.

negative reward received from being too close to dangers. One of the experiments was to determine an acceptable value for γ_{ot} empirically. γ_{ot} was in that experiment varied between 0 and 5 and the result is shown in Figure 5.8. The results indicate that even the γ_{ot} parameter can have a wide range of possible values if just 0 is avoided. This is a quite surprising but positive result because it demonstrates that no detailed tuning of γ_{ot} is necessary.

Temporal Validity Factor

An abstraction is always considered invalid after a certain time T_{replan} (see Procedure 5.7). T_{replan} is calculated by $\alpha_T T_{horiz,hsn}$ where α_T is called the *temporal validity factor*. Figure 5.9 shows the result when α_T varies between 0 and 0.4. The best result seems to be when α_T is set to approximately 0.075 but acceptable results are received for values between 0.1 and 0.3 as well and the performance seems to gradually drop after that.

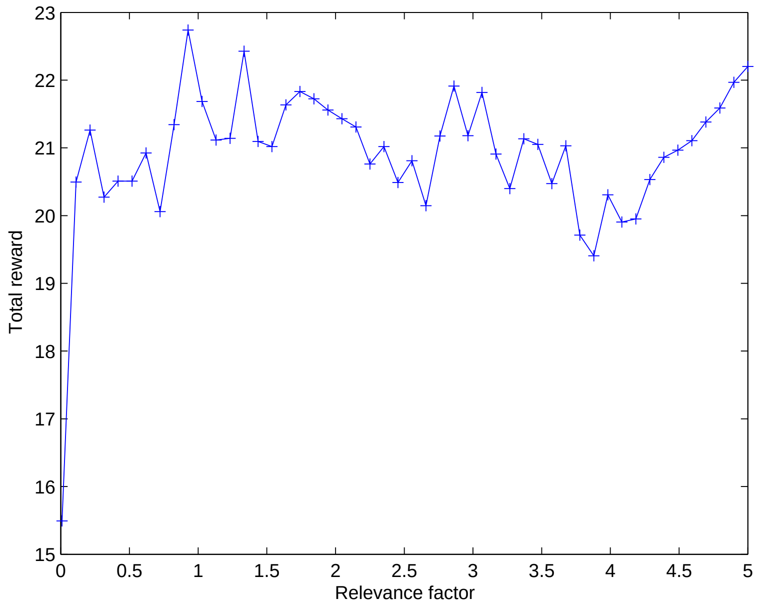

Figure 5.8: The result when γ_{ot} is varied.

Abstraction Levels

There are many ways to test the benefits from using more abstraction levels. One such method is to increase the minimum cell size, making the refinements occur less frequently. Another way is to simply set a maximum number of abstraction levels. Figure 5.10 shows the result when the maximum number of abstraction levels are varied. The result verifies that several abstraction levels are beneficial, even when the dynamics of the environment penalizes the extra computation. The figure also demonstrates the effect of the cutoff condition which prevents that more than three abstraction levels are created.

Architecture Speedup

A simple experiment was performed when the computational resources was decreased 10 times and increased 100 and 1000 times the original. The result is shown in Table 5.1. The reason for the large gap between S_{max} in the two experiments was that the number of actually visited states during the solution phase was much lower than 100000. The speedup of 100 gives a rather large step in the result (from approximately 21 to 36).

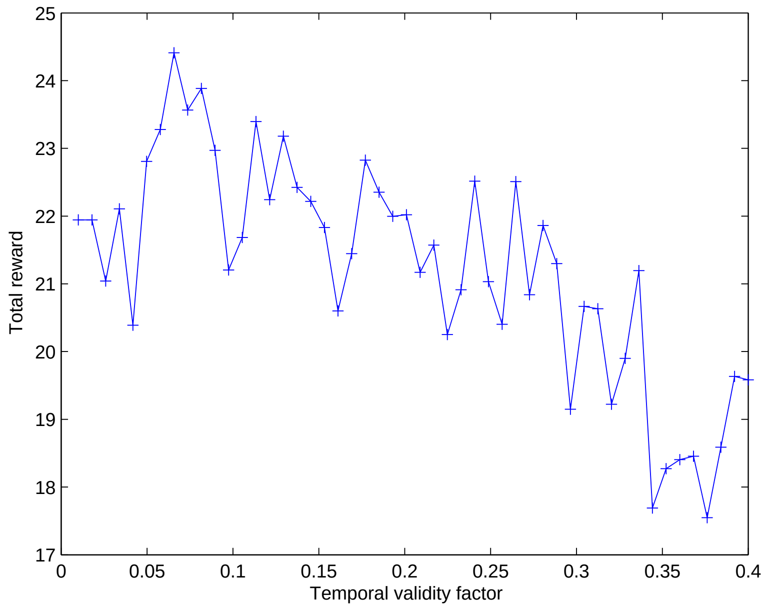

Figure 5.9: The result when the temporal validity factor α_T is varied.

Equal Relevance

It is interresting to investigate how effective the dynamic abstraction is compared to a fixed abstraction. The problem with such an experiment is that the curse of dimensionality makes the state space explode when many external agents are present. Suppose that the agent represents every external agent with n discrete values in the discretization. If there are N external agents in the environment and the agent represents its own position with N_p number of discrete values, $|S|$ becomes $N_p \cdot n^N$. When N is equal to 10 (the highest number of external agents in the tests), by just setting n to 9 discrete values (which could be used to construct a 3x3 grid or use clusters) and N_p to 16 (possibly a 4x4 grid) leads to a state space larger than 55 billion. This size of S can actually be dealt with if function approximation methods are used or symmetries of the problem are exploited. In this way it is possible to construct a fixed abstraction policy that can be used to possibly outperform the dynamic abstraction method. But what if just one or two external agents are added or removed? With this approach, the state space will become totally different and the solution policy may turn out to be useless. The point is that if it is possible to know exactly how the state space will look like (as in backgammon or chess), it is possible to construct

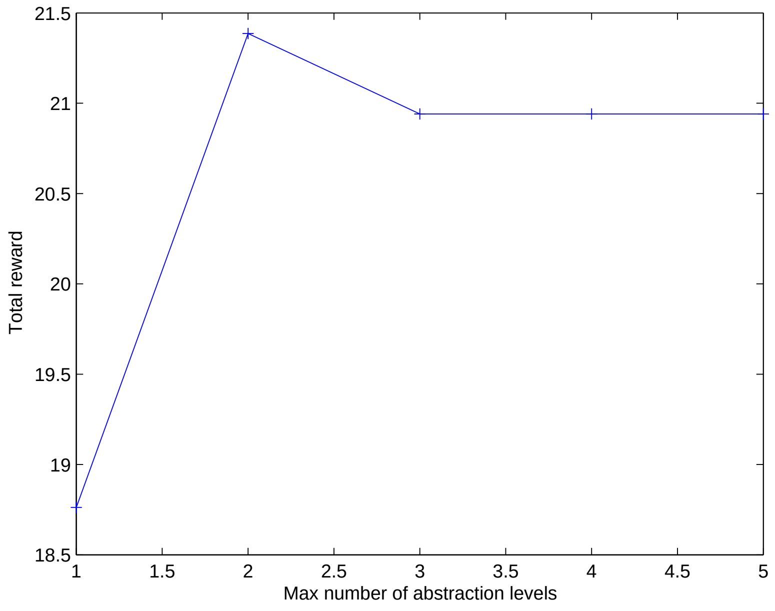

Figure 5.10: The result when the maximum number of abstraction levels are varied.

high quality solutions. In a more flexible environment where objects can be added or removed in a mixed-initiative setting, the dynamic abstraction method is probably more effective, even if only small problems can be solved at a time.

An experiment was performed when the relevance for all features was set to an equal value. The resulting mean reward for the equal relevance was only 4.26. When the relevance information was used, the mean reward received was 20.94 instead which demonstrates the importance (in this task environment) of constructing the abstractions dynamically with a reasonable relevance measure.

5.6 Comments

This chapter has presented an implementation of the DARE method for the fully observable UAS environment class. The experiments only make comparisons between different settings of the parameters in the dynamic abstraction, but, by observing the behavior of the agent many decisions it makes are very reasonable. In some cases, the agent makes greedy decisions

Speedup	S_{max}	Sim. steps	Mean reward
0.1	50	150	4.89
1	150	1000	20.94
100	1000	20000	36.38
1000	100000	100000	40.16

Table 5.1: Three results when the architecture was slowed down 10 times and sped up 1, 100, and 1000 times the original speed.

where it takes an easy way out by finishing at a nearby finish area when there are still unclassified observation targets in the area that are difficult to reach due to dangers.

The implementation performs well and scales up nicely when the number of objects in the environment class increases due to the dynamic abstraction. It has been demonstrated how all parts of DARE can be implemented in practice and possibly the most interesting part is the implementation of the dynamic abstraction which actually performs an optimization over the state distribution in the discretizations.

It takes some effort to implement DARE in a given environment class and it may be an overkill in this particular case. It is expected that most of the benefits of DARE will be demonstrated in more complex environment classes where many different types of abstractions and planning model types have to interact and it is too time-consuming to specify beforehand how the agents should view the different parts of the environment in every case. Figure 4.2 on page 42 illustrates one vision where different planning model types and abstractions can be used at the same time depending on whether they are suitable to represent the abstractions sufficiently well.

The experiments strongly indicate that the choice of abstractions should depend on the available computational resources when the task environment is dynamic (see e.g. Table 5.1). For planning with a performance measure, this tradeoff between accuracy and feasibility can actually be tested when the experiments take the deliberation time into account through the model with simulated dynamics.

Although the fully observable UAS environment class is dynamic, continuous and stochastic, it is actually not of much use for any realistic mission for the UAS Tech system, mainly due to the assumtion of full observability but also because of the simplified movement assumption and that no obstacles exist. The next chapter will describe a case study where some of the ideas of DARE are applied to a partially observable extension of the fully observable UAS environment class.

Chapter 6

Case Study II

Chapter 5 described an implementation of the DARE method which demonstrated how dynamic abstraction can be performed in practice and providen an example of problem generation and abstraction monitoring.

The environment class used in that case study was rather simple. The assumptions of a fully observable task environment, no obstacles and a very simple movement assumption (kings moves) makes it impossible to use directly for a mission for the UAS Tech system. The environment class can still be difficult to handle without a dynamic abstraction mechanism due to the curse of dimensionality (if MDPs are used as planning models).

Since one of the goals of the work with the DARE method is to push dynamic abstraction techniques into realistic settings, the next step is to implement it for a more realistic environment class, which is the focus of this chapter. This new environment class still has dangers and observation targets as in the previous case study, but now these external agents are only partially observable and the environment contains obstacles that must be avoided.

In this environment class (described further in Section 6.1), partial observability means that the agent can not see through the obstacles with its noisy sensor (a camera in this case) and not further than a certain range.

Another modification of the environment class from Chapter 5 is that the agent is restricted to move on linear path segments that, in this case, are returned by a roadmap-based pathplanner [61].

DDNs are still used to model the task environments, but in this case observation variables are used as well and filtering is necessary to keep track of the external agents. Particle filters are used because of the apparent need to model the multi-modal and nonlinear characteristic of the probability distribution.

The DARE method's approach of dynamically generating planning models "on the fly" is also followed in this case study. The main difference between the method used in Chapter 5 is that the belief state is used to generate problems and the type of planning model is different; an adapted version of depth-limited lookahead is used instead of reinforcement learning. Optimization is still used, but in this case the result of the optimization determines the possible points that the UAS agent will consider flying to instead of the state distribution.

All the features in the DARE method are not implemented e.g. dynamic abstraction hierarchies. The implementation uses two fixed levels at all times; one for the flight manouvers and one for the detailed camera movement. It is still considered as dynamic abstraction since the planning models are generated depending on the current situation.

The results presented in this chapter are published in [58].

6.1 Task Environment Class

Figure 6.1 shows an instance of the *partially observable UAS environment class*.

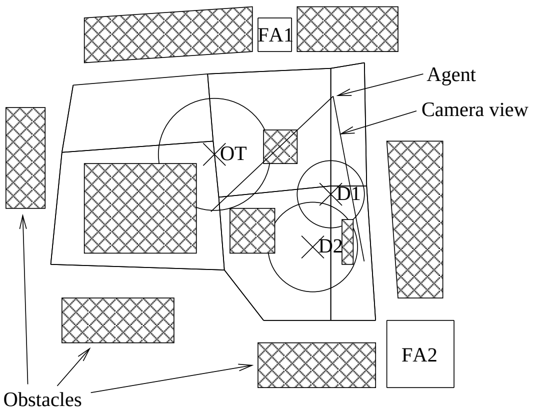

Figure 6.1: An instance of the partially observable UAS environment class. D1 and D2 are dangers and OT is an observation target. The circles around the external agents show the cost radius and the maximum classification distance. FA1 and FA2 are finish areas.

In this environment class, the agent is only allowed to move on linear segments that are returned from a pathplanner. The pathplanner in this case is roadmap-based which is quite similar to one of the planners that are used in the UAS Tech system [61] where a probabilistic roadmap planner is used. The main difference is that the environment is in this case two-dimensional and the roadmap is generated deterministically by setting the vertices to the surrounding points of the obstacles and then connecting every visible vertex. Plans are generated by connecting the start and goal vertex to the roadmap and performing A* search with the straight-line heuristic. Figure 6.2 illustrates an example of a path from A to B generated by the pathplanner.

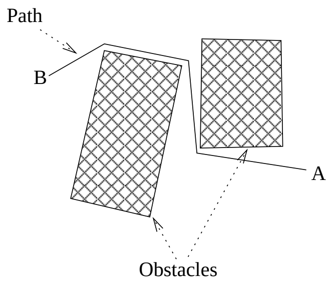

Figure 6.2: The figure illustrates a planned path from point A to B, returned from the 2D roadmap-based pathplanner.

The UAS agent is equipped with a camera that can be used to track the external agents and classify the observation targets which can be viewed as a more detailed sensing action where more information is extracted from the target than its position. It is assumed that it has a maximum range and that some kind of geographic information system (GIS) is used to map the image tracking information to world coordinates. This type of functionality is implemented in the UAS Tech system where a Kalman filter is used to keep track of the UAS's pose with the input given from the inertial navigation system and GPS [11].

The rewards that are received still depend on the distance to dangers and successful classifications as in Chapter 5, but in this partially observable environment the external agent has to be visible from the agent to modify the (positive or negative) reward.

6.2 Task Environment Model

Figure 6.3 on page 72 shows a DDN for an instance of the partially observable UAS environment class with one freely moving danger and one road network bound observation target. Every freely moving external agent is simulated by a DBN similar to the one used in chapter 5 with the help of a random number generator. The main difference is that the freely moving external agents try to avoid the obstacles in the environment as well, which is performed with a simple collision avoidance technique that makes the external agent slow down when it is about to hit an obstacle.

The DBNs for road network bound external agents are identical to the DBNs used in the fully observable UAS environment class.

The main difference from the fully observable environment class is the presence of observation variables that are used to model the noisy sensor of

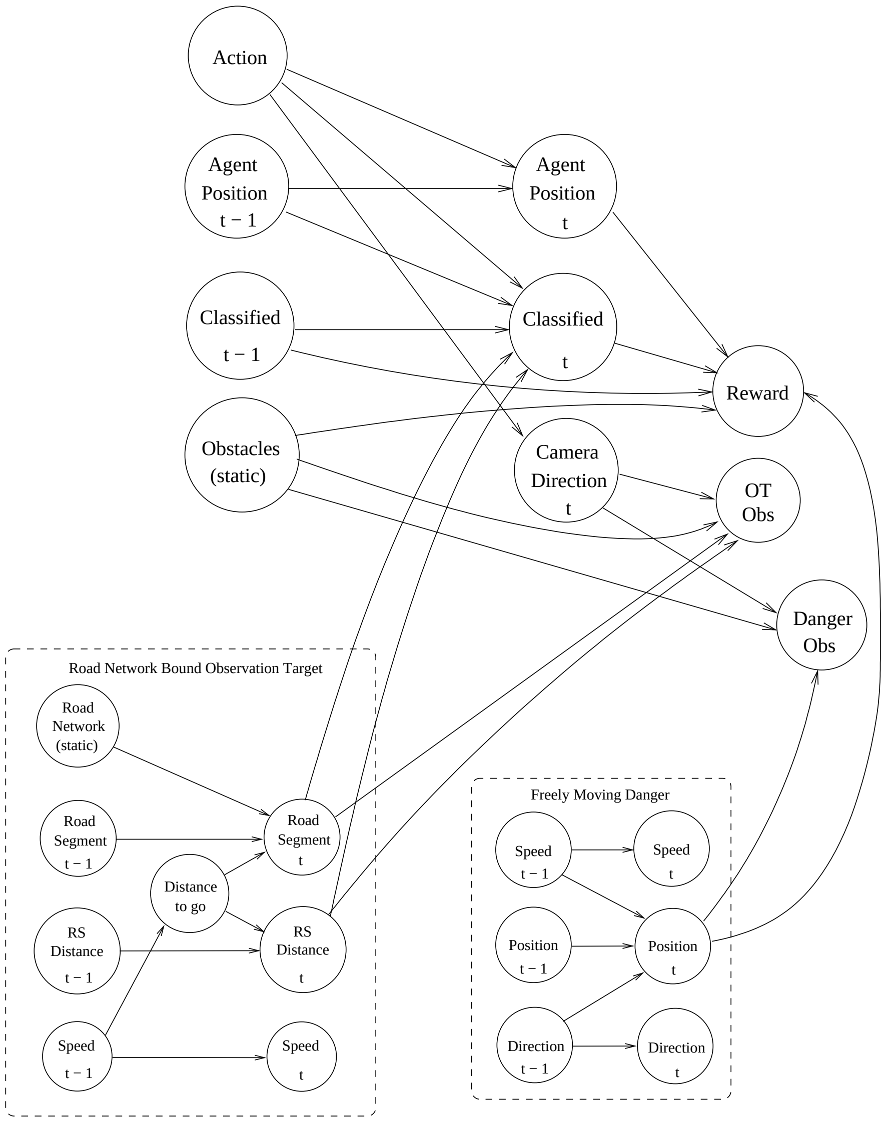

Figure 6.3: An example of a DDN for a task environment that contains one road network bound observation target and a freely moving danger.

the agent. There is one observation variable O_{ea} for each external agent ea which has a domain of $\mathbb{R}^2 \cup NO$ where NO indicates that ea was not observed at all. In Figure 6.3, these variables are called *OT Obs* and *Danger Obs* for clarity.

Since particle filters (see Section 6.4) are used to represent the belief state, likelihood functions that are proportional to $f_{Obs_{ea}}(o|X)$ have to be constructed for the observation variables where X is the set of state variables that describes the agent's position, ea's position (see Section 6.4) and whether ea is within line of sight and within the camera's view area. Figure 6.4 on page 73 illustrates the four different cases that are considered in the observation model.

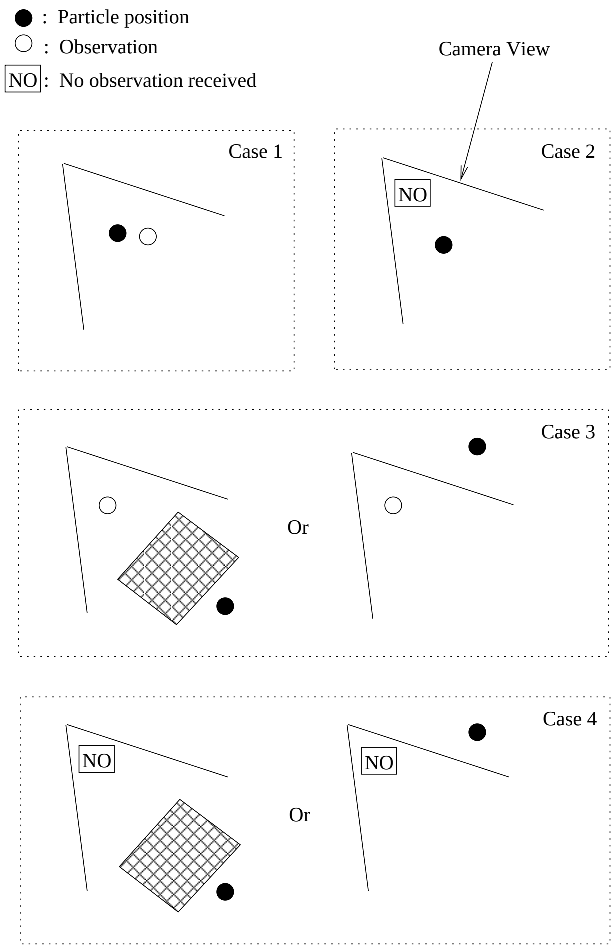

Figure 6.4: The four different cases in the observation model. Case 1: An observation was recieved and the particle is actually within the camera view. Case 2: No observation was received even if the particle is visible and within the camera view. Case 3: A spurious observation was received. Case 4: No observation was received when the particle either is outside the camera's view or blocked by an obstacle.

Case 1 in Figure 6.4 is handled by a gaussian likelihood function $L_{obs,v}(d)$ with zero mean and a 5 meter standard deviation where d is the distance from the observation to the external agent, *obs* means that an observation was received and v that the external agent actually was visible. Cases 2, 3 and 4 are handled by three different constant likelihoods $L_{no-obs,v}$, $L_{obs,v}$ and $L_{no-obs,\neg v}$ to model the possibility of spurious observations.

6.3 Skills

The available skills in the partially observable UAS environment class match the actual Task Procedures that are available in the UAS Tech system much better than in the fully observable one. The agent has skills that can perform the following actions:

- **Fly path:** This skill can fly a given linear path with a constant speed and it stops at the target position. It can be interrupted at any time.
- **Wait:** This skill corresponds to the UAS Tech system's hovering action. A simplifying assumption is that the agent can stop immediately without any delay.
- **Turn camera:** A skill that can turn the camera in any direction. It is assumed that the camera can be instantaneously turned from one angle to another and that the skill can be executed in parallel with all the other skills in the system.
- **Finish:** This skill corresponds to the UAS Tech system's automatic landing action. The agent must be within a finish area to be able to execute that skill.

It is also assumed that in parallel, the agent performs automatic classification and tracking of the external agents which could be considered as another skill that is continuously executing.

6.4 Belief State and Filtering

The belief state of the agent is represented with a set of particle filters, one for each external agent. It is assumed that the number and types of external agents are known in advance.

The particle filter uses particles to represent a probability distribution and the particles are different depending on what type of external agent it is. A road network bound external agent's particle includes the current road segment, distance travelled on that segment and the current velocity. A freely flying agent instead has the current position and velocity (both 2-dimensional) in each particle.

The sequential importance resampling (SIR) [3] algorithm is used to update the belief state after each step. The SIR algorithm is shown in Procedure 6.1 and it uses a so called *low variance resampling* algorithm (see Procedure 6.2).

Every particle filter for the external agents is updated with a separate call to SIR() to update the agent's full belief state. x_k in SIR() is then the set of all state variables in the external agent's DBN (see e.g. Figure 6.3) and x_k^i is the i:th particle in the current particle set $X_{ea,k}$. $Obs_{ea,k}$ is the observation state variable that was observed which is set to a position or NO (no observation).

Procedure 6.1 SIR($X_{ea,k-1}$, $Obs_{ea,k}$)

```
1:  for i = 1 to $N_s$ do
2:      Draw $x_k^i \sim P(X_k|x_{k-1}^i)$
3:      $w_k^i \leftarrow L_{Obs_{ea,k}, v \in x_k^i}(d)$
4:  end for
5:  $t \leftarrow \sum_i^{N_s} w_k^i$
6:  for i = 1 to $N_s$ do
7:      $w_k^i \leftarrow \frac{w_k^i}{t}$
8:  end for
9:  Return LOWVARIANCESAMPLE($w_k$, $x_k$)
```

Procedure 6.2 LOWVARIANCESAMPLE(w_k, x_k)

```
1:  $c_1 \leftarrow 0$
2:  for $i \leftarrow 2$ to $N_s$ do
3:      $c_i \leftarrow c_{i-1} + w_k^i$
4:  end for
5:  $i \leftarrow 1$
6:  $u_1 \sim \mathrm{U}[0, N_s^{-1}]$
7:  for $j \leftarrow 1$ to $N_s$ do
8:      while $u_j > c_i$ do
9:          $i \leftarrow i + 1$
10:     end while
11:     $x_{k,res}^j \leftarrow x_k^i$
12: end for
13: Return $x_{k,res}$
```

Figure 6.5 shows an example of a belief state that is represented with particle filters in the partially observable UAS environment class. The true state is illustrated in Figure 6.1. The agent has localized the two dangers D1 and D2 quite well but is still uncertain of where the observation target OT is. The figure shows the particle filter's capability of modeling the multi-modal characteristic of the probability distribution that seems to be useful in this environment class.

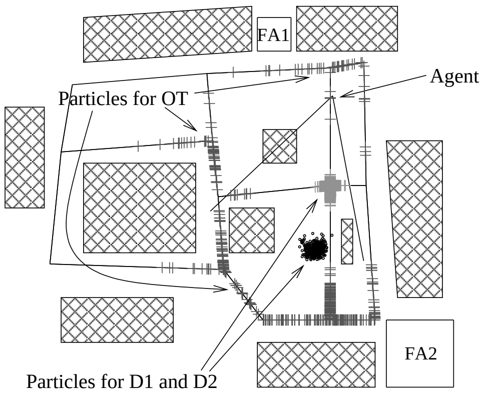

Figure 6.5: An example of a belief state represented with particle filters. The agent has localized the two dangers D1 and D2 but not the observation target OT.

6.5 DARE Implementation

This section describes the partial implementation of DARE in this environment class. It is only a *partial* implementation due to the fact that the number of abstraction levels does not change dynamically depending on the situation. It is always assumed that the movement of the agent can be planned without taking the detailed camera movement into consideration. The camera movement is determined after the agent knows what direction (if any) it should go.

The implementation of DARE for this environment class still makes use of the dynamic view of planning models. New planning models that can be used for depth-limited lookahead, are generated depending on the current belief state of the agent and implements the generateProblem() procedure in DARE. The depth-limited lookahead implements the solve() procedure in DARE.

Section 6.5.1 presents the problem generation procedure and Section 6.5.2 describes the details of how the depth-limited lookahead is performed.

6.5.1 Planning Model Generation

The planning model generation consists of the following two steps:

- Point selection and
- Connection of the selected points with a pathplanner

Point Selection

The first step of the planning model generation procedure is to find a set of good points that the agent should consider flying to. A good point should be close to unclassified observation targets and sufficiently far from dangers. The number of points that are selected, N_{pg}, should not be too many (which would make the model too big) or too few. N_{pg} is an important design parameter when the tradeoff between accuracy and feasibility of the planning model is considered for a certain architecture (see Section 6.6).

The problem of selecting the set of points is formulated as an iterative optimization problem. A utility measure is defined for a point given the current belief state and this measure is used to compare different points. One point at a time is selected and the previously selected points are used to modify the utility function for increased diversity of the points (otherwise the same point can be selected over and over). The positions of dangers and observation targets contribute to the utility but also the distance from the selected point to the agent and whether the point is within a finish area or not matters.

The belief state is represented with particle filters and each particle in every filter contributes to the total utility. This yields some kind of approximation of the *expected utility* of a selected point.

All the contributions from dangers, observation targets, finish areas, distance from the agent and previously selected points are added and specify the total expected utility of selecting that particular point.

The utility contribution from a danger's particle, U_{do}, depends on the distance d from the particle to the agent:

$$\begin{cases} U_{do} = min(-C_{max} + \frac{C_{max}}{C_R} d, 0) \text{ if } do \text{ is visible from the agent} \\ 0 \text{ otherwise} \end{cases} \tag{6.1}$$

where C_R is the cost radius of the danger.

Similarly, the point utility for observation targets also depends on the distance but one also needs to consider whether it has been classified previously or not:

$$U_{ot} = \begin{cases} R_{cl,ot} - \frac{R_{cl,ot}}{d_{ot,max}} d & d < d_{ot,max} \text{ and } \neg cl_{ot} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

where $R_{cl,ot}$ is the reward for classifying the target ot, $d_{ot,max}$ is the maximum classification distance and cl_{ot} the boolean variable that specifies whether ot has been classified previously or not. Notice that observation targets that have been previously classified provide nothing to the point utility, making it possible for the agent to focus on more important external agents or finish areas.

The utility contribution U_{fa} from a finish area fa is the same as the corresponding finish reward if the point is within fa and the agent has not executed the finish action yet (which is only important when U_{fa} is used during the depth-limited lookahead. See Section 6.5.2).

To provide a simple way to create diversity of the selected points, the point selection takes the previously selected points into account. The penalty function $U_{\mathbf{p}}$ is used for the set of previously selected points $\mathbf{p}$ that depends linearly on the distance between the considered point with one exception: if the newly selected point is located within a finish area and no other point is, no penalty is given.

The distance from the considered point to the agent also contributes to the point's utility. The cost of travelling in a straight line from the agent to that point is used as the cost estimate.

Figure 6.6 shows an example of the utility function used for point selection when the belief state is the one in Figure 6.5.

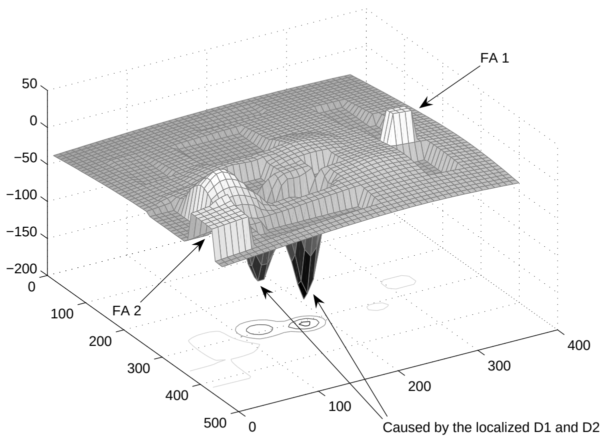

Figure 6.6: An instance of the utility function for the point selection optimization problem for the belief state shown in figure 6.5. The plot shows the utility function when the first point is selected. The current position of the agent is always added to the previously selected points. Notice the negative utility caused by the localized dangers.

Since it is rather expensive to estimate the expected utility of points, a local search algorithm (see Section 2.4) is used to select the points that are then used to generate a planning model (see Section 6.5.1).

Graph Generation

When the set of points have been selected, paths are planned between every combination of distinct point pairs to construct a graph. This operation is performed with a roadmap-based 2D pathplanner using A*-search which is quite similar to the PRM-based pathplanner that is used in the UAS Tech system [61]. The set of resulting paths determines the finite set of actions that the agent can perform, making it possible to perform lookahead-based planning in the partially observable UAS environment class.

The number of paths is reduced to lower the branching factor for the depth-limited lookahead which in this case means that paths that contribute little or nothing are removed. This is performed by considering all triples of distinct points. If the length of a path between two points a and b is given by $l_{a,b}$, then if $l_{a,b} + l_{b,c} < \alpha_l \cdot l_{a,c}$, the path from a and c is removed from the planning model. α_l is set to 1.1 in the implementation.

Figure 6.7 shows an example of how a dynamically generated planning model can look like given the utility function in figure 6.6. Note that some of the paths are going straight through the positions of the dangers, which seems to be very irrational. The point selection does not take the path to the points into consideration and therefore these seemingly stupid paths arise. The "stupidity" of those choices is discovered later during the depth-limited lookahead because if the agent simulates such a path, a large negative reward will be received.

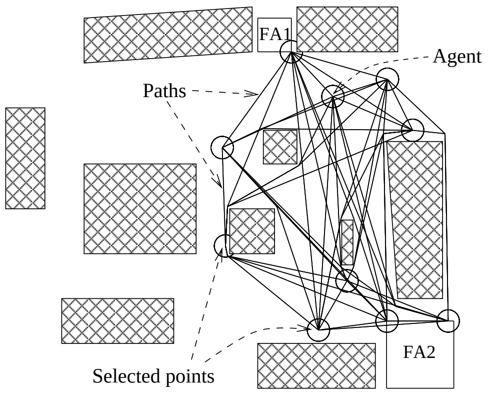

Figure 6.7: A generated planning model for depth-limited lookahead given the point value function illustrated in figure 6.6. The selected points are drawn with circles. The road network is hidden for clarity.

6.5.2 Solution Method

The planning in the planning model is done by a depth-limited lookahead from the current belief state (which is represented by the particle filters). The planning model is not directly suitable for applying the depth-limited lookahead idea since by simply considering the movement from one point to another as a primitive action without modification ignores all the possible observations and rewards that are received *during* the execution.

Procedure 6.3 shows the depth-limited lookahead algorithm that is used in the implementation. It enumerates all actions as normal depth-limited lookahead does but samples N_{obs} sequences of observations and belief states instead of enumerating the (infinite) number of possible observations during the execution of the action.

During lookahead, the camera is assumed to have a 360 degree field of view which means that the movement of the camera is planned at a later stage. If the movement of the camera would be considered in DEPTHLIMITEDLOOKAHEAD(), the branching factor would become too large and that abstraction would not be a good choice.

Procedure 6.3 DEPTHLIMITEDLOOKAHEAD(depth, BS_{cur})

```
1: if depth $\geq d$ then
2:   Return $\langle \frac{\sum_{x \in BS_{cur}} U(x)}{|BS_{cur}|}$, Wait$\rangle$
3: end if
4: $BS_{start} \leftarrow BS_{cur}$
5: bestValue $\leftarrow -\infty$
6: bestAction $\leftarrow$ Wait
7: for all Actions $a$ possible in $BS_{cur}$ do
8:   sum $\leftarrow 0$
9:   for $i \leftarrow 1$ to $N_{obs}$ do
10:     Filter a sequence of steps starting with $BS_{start}$
11:     Store belief state result $BS_{end}$ and reward $r$
12:     $\langle U_{est}, a_{best} \rangle \leftarrow$ DEPTHLIMITEDLOOKAHEAD(depth + 1, $BS_{end}$)
13:     sum $\leftarrow$ sum $+ r + U_{est}$
14:   end for
15:   $U_{est}(a) \leftarrow \frac{\text{sum}}{N_{obs}}$
16:   if $U_{est}(a) >$ bestValue then
17:     bestValue $\leftarrow U_{est}(a)$
18:     bestAction $\leftarrow a$
19:   end if
20: end for
21: Return $\langle$bestValue, bestAction$\rangle$
```

When a movement action has been selected by the depth-limited lookahead, the solution in the HSN consists simply of the **bestAction** returned from DEPTHLIMITEDLOOKAHEAD(). This solution is then refined to select the camera movement.

6.5.3 Camera Movement

The DARE method use the createSubproblems() procedure to refine solutions. The implementation of createSubproblems() in the partially observable UAS environment class is fairly trivial since it is always the case that the camera movement is planned when a solution action is returned by depthLimitedLookahead(). The refinement does not have to be fixed like that. It can be defined by generating a new DDN which represents the subproblem of performing the specified action such as moving towards a created finish area (like in Section 5.4.4) but that has not been implemented.

The camera movement, given the action returned from the depth-limited lookahead, is planned by enumerating a set of possible camera directions and selecting the one that maximizes the *expected relevance* of the visible particles in the belief state. The relevance $R_{p_{do}}$ for a visible danger object particle p_{do} is calculated by $C_{max} \cdot e^{-\alpha_{p_{do}} \cdot d}$ which is the same measure used in Section 5.4.2 when the state distribution was determined. The main differences are that the point to point visibility and the camera's view area are taken into account and that it is the expected relevance that is calculated with a contribution from all particles.

Since the camera is assumed to be capable of pointing instantaneously towards a selected point, independent of its previous angle, the current belief state BS_{cur} is used directly to select the camera angle.

6.5.4 DynabsSolve Implementation

All the parts of the implementation of DARE's dynabsSolve() procedure have been presented and are in this section listed in Procedures 6.4 and 6.5. What is missing for a full implementation of DARE is the dynamic generation of abstraction levels. This can be done in many different ways by creating sub HSNs that consider the first step of the solution as a subproblem. For example, a subproblem to a "fly path" action can view the target position as a finish area and try to find a better way to get there than the previously planned path. On that level of abstraction, it might also be possible to use more detailed action descriptions that e.g. consider the velocity of the agent.

Procedure 6.4 dynabsSolve(hsn, TEModel)

```
1: points(hsn) ← findPoints(TEModel)
2: problem(hsn) ← createProblem(TEModel, hsn)
3: action(hsn) ← depthLimitedLookahead(d, BS(TEModel))
4: timestamp(hsn) ← currentTime()
5: planCameraMovement(hsn)
6: Update movement skill according to action(hsn)
```

Procedure 6.5 PLANCAMERAMOVEMENT(hsn, TEModel)

```
1: bestAngle ← FINDBESTANGLE(BS(TEModel))
2: Update camera skill with bestAngle
```

6.5.5 Replanning

A solution to a planning model is only kept for a certain time, as in Section 5.4.5 where a timestamp was used to keep track of when a solution should be considered outdated. In the partially observable UAS environment class, two different temporal horizons are used; one for the movement solution and one for the camera direction.

Since the camera movement is relatively easy to compute, that solution is considered outdated at every iteration of the REPLANIFNECESSARY(). The agent movement is replanned repeatedly every T_r second and the default setting for T_r is 2 seconds.

Procedure 6.6 REPLANMOVEMENTIFNECESSARY(hsn, TEModel)

```
1: if (CURRENTTIME() - timestamp(hsn)) > $T_{replan}$ then
2:    set subNode(hsn) to nil
3:    DYNABSSOLVE(hsn, TEModel) {Replan}
4: else
5:    PLANCAMERAMOVEMENT(hsn, TEModel)
6: end if
```

6.6 Experiments

A set of experiments have been performed with the implementation in order to show some of the tradeoffs between accuracy and feasibility when deliberation time is considered.

The following parameters were varied in the experiments:

- Number of points that are selected during the planning model generation, N_{pg}
- Depth for the depth-limited lookahead, d
- Number of sampled observation sequences for the depth-limited lookahead, N_{obs}
- Replanning period T_r
- Number of particles used for belief state during depth-limited, N_{pfs}
- Number of particles used for belief state during point selection, N_{pl}
- Whether simulated dynamic mode is used, SD (see Section 6.6.1)

N_{pg}	d	N_{obs}	T_r	N_{pfs}	N_{pl}	SD	Value
7	1	8	2.0	5	5	Yes	48.02

Table 6.1: The default configuration.

6.6.1 Setup

Most of the experiments were performed in simulated dynamic mode which, as in the fully observable case, means that the environment is evolving during the agent's deliberation. The deliberation time is in this case estimated by counting the most frequently and costly operations that are performed during point selection and planning. The two operations that are used for deliberation time estimation are the utility calculations of a point during point selection and planning, and the simulation step of the DDN that is used for prediction during planning. The time for those operations were first measured in the implementation and then assumed to be fixed during the experiments (for the purpose of assuring repeatability).

6.6.2 Results

Since it is not feasible to generate results for every possible configuration of the parameters described previously, some configurations were tested that point out interesting behavior of the implementation. First a *default* configuration was created, with some trial and error, which is shown in Table 6.1 together with the resulting value. The value is equal to the mean sum of rewards that are received during 500 test runs. The default configuration is used as the basis for the experiments when a subset of the parameters are changed.

Number of Particles

One of the experiments was to investigate what happens when the number of particles used during the lookahead and point selection are varied. Since the dynamics of the environment is simulated, deliberation time is penalized both by the cost of waiting during planning but also with increased response times in dangerous situations. The question is where the optimal (static) tradeoff between accuracy and feasibility is (with the N_{pl} and N_{pfs} parameters) given the simulated deliberation penalty. The result of the experiment is shown in Figure 6.8 which demonstrates the importance of taking the dynamics and available computational resources into account. The best result was obtained when N_{pfs} was set to 2 and N_{pl} to 4 which was much lower than expected.

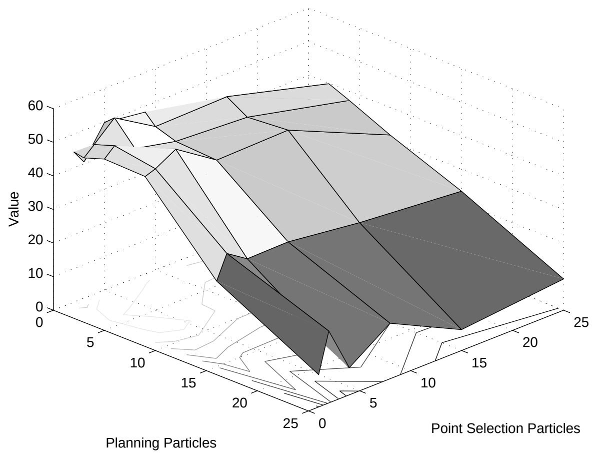

Figure 6.8: The result when the number of particles for point selection and planning are varied.

Lookahead Depth and Observations

The default configuration uses a lookahead depth of 1, which is rather extreme. But the best results were in fact obtained when this setting was used. Table 6.2 shows the result of an experiment when the lookahead depth and number of observation samples are varied simultaneously. The result clearly indicates that a lookahead depth of 1 should be used for this environment class when the computational resources are taken into account. The best lookahead depth also highly depends on the planning model generation, which in this case generates models with very long temporal steps.

No Simulated Dynamics

Some tests were also performed when the simulated dynamics was disabled. Table 6.3 shows the three different configurations that were used together with their corresponding results. The results are clearly better than the best result when simulated dynamics is used (56.88) but as the table shows, the number of points selected and particles used are much higher and it requires a lot more computation.

6.7 Discussion

This chapter has presented a partial implementation of the DARE method in more realistic environment class than in Chapter 5 which includes partially

N_{obs} \ d	1	2	3	4
1	53.58	18.88	-247.2	-2502
2	53.71	-3.932	-2137	NA
3	56.88	-64.09	NA	NA
4	53.56	NA	NA	NA
5	56.43	NA	NA	NA
6	52.24	NA	NA	NA
7	49.31	NA	NA	NA
8	48.02	NA	NA	NA

Table 6.2: The results when the number of observations and lookahead depth parameters are varied.

N_{pg}	d	N_{obs}	T_r	N_{pfs}	N_{pl}	SD	Value
10	1	15	1.0	50	50	No	59.39
20	1	15	1.0	50	50	No	62.46
20	1	20	1.0	200	200	No	64.30

Table 6.3: Three results when no simulated dynamics is used.

observability and obstacles.

The experiments have demonstrated that it is important to take the available resources into account when creating planning models dynamically.

All features of DARE have not been implemented. No dynamic abstraction hierarchies are created. The author belives that this is probably more useful when the environment is more complex and can e.g. include arbitrary 3-dimensional building structures and more complex models of the external agents or is part of the real world. Such environments would require submodels on different abstraction levels and the dynamic abstraction would probably be more useful than it currently is for this task environment class.

For complex environment classes, dynamic abstraction should be performed during the filtering as well. It would then be possible to focus on more relevant objects at the moment but it is also important to be able to backtrack in the "model space" if a previously considered irrelevant object suddenly becomes relevant. The agent can then change its models depending on this new information but it may also have to update a belief state where this new object is considered. This is a good example where a memory of previous percepts and actions can be used to update the current belief state by "refiltering" with this new belief state definition.

Chapter 7

Conclusion

This thesis has investigated the consequences of using a more dynamic view of planning models than traditionally proposed within Artificial Intelligence. It has been argued that dynamic abstraction is a suitable tool for planning in more open-ended environments where planning models can be generated dynamically. The use of dynamic abstraction for planning leads to the problem of monitoring the different abstractions continuously and performing model reconstruction and replanning when necessary. This methodology is captured in the DARE method that was presented in Chapter 4.

Two partial implementations of DARE have been demonstrated where planning models have been generated depending on how important different aspects of the environment have been judged. Chapter 5 presented an implementation for a fully observable task environment class where dynamic abstraction hierarchies were implemented and the planning models were generated dynamically depending on how important the different features in the environment were. Chapter 6 illustrated how some ideas of DARE were implemented for a more realistic, partially observable task environment class.

7.1 Future Work

Some very specialized methods to perform dynamic abstraction have been used in this thesis. What steps can be taken to generalize these methods? One possible step is to try to extend the case studies (especially the second one) to make it work in a real robotic system e.g the UAS Tech system. The task environment class would then be a part of the real world which introduces many problems. Future work related to this approach will be discussed in Section 7.1.1.

Another approach is to investigate what type of high level reasoning is necessary to draw conclusions about what abstractions to use. Such an investigation could lead to some kind of theory of abstractions which can be used to perform more general dynamic abstraction methods. This approach will be discussed in Section 7.1.2 where the main focus is to develop such a theory to make it possible to generate task environment models depending on relevance information given a certain task and beliefs.

7.1.1 Extentions to the Case Studies

The second case study was the most realistic implementation discussed in this thesis and was part of the author's intention towards the use of DARE for real missions with the UAS Tech system.

A few things must be improved in order to use that solution. First of all, the implementation must be extended to work in 3D which means that the point selection and model generation has to work with one more dimension, which is straightforward in theory but could be a problem in practice due to the increased complexity. The path planner that already is used in the UAS Tech system could be used to generate the set of paths needed to formulate the planning model.

The implementation in the second case study simply stopped the agent during the planning model generation and lookahead which is probably not feasible during a real mission due to the extra delay introduced when a helicopter system has to perform a breaking manouver before each decision. A more efficient solution is to perform the planning model generation and lookahead *during* flight instead which means that the model generation should start with a prediction of the future belief state where and when the solution will be used. To be sure that this time limit is followed (the underlying control system demands that the next path segment is sent before a certain time), one could try an idea where the complexity of the planning problem is iteratively increased. Suppose that it starts with a very simple problem formulation with just a few points. Planning is performed and a possible decision is ready to use. If there is time left, a more complex planning model can be constructed with more points and possibly more particles (one can also reuse the old points).

There is also a lot of work ahead with a more realistic sensor model and the actual detection and tracking of vehicles. Work is already underway to make it possible to detect vehicles with a combination of infrared, color and feature input.

Another extension to the second case study is to make it possible to focus the attention of the agent with different number of particles for the representation of external agents. The same kind of search used in the first case study can be used to select a reasonable distribution of particles given the available time and computational resources.

A different type of planning method can also be tested in the future where only one target point is considered at a time. Suppose that a point is greedily selected by the point selection method and a path to that point is planned with the path planner. Then that path is simulated a few times as used in the task planning phase and a set of the rewards and time points during the path are stored or generalized by the examples. Then a new point is selected that takes these rewards and time points into account and a path is planned where the path planner takes the previously rewards into account. This goes on until time runs out. The nice thing about this approach is that the path planner gets feedback from the task planner all the time. The question is then how the resulting rewards are stored, but a simple instance-based learning method could be a start.

7.1.2 Dynamic Task Environment Models

Both of the case studies used task environment models in different ways in order to perform task planning. These models were rather fixed at a particular level of abstraction such as the representations of the external agents and the road network were fixed. The step size could be varied which makes it possible to take fewer and larger steps.

The task environment models determine to a large extent the level of abstraction for the task planning if they are used in this way. It would therefore be interesting to try and generate task environment models with the help of relevance information given that a certain task should be executed by an agent in a particular situation. This is particularly important when the task environment is part of the real world.

In [43], Levy et al. automatically generated models that describe physical dynamical systems depending on what is considered relevent given a certain query. The generated models are complex enough to answer the query but also as “simple as possible” given a developed theory of relevance. Logic programming was used to reason about possible models given the query as a goal statement.

Could then the same procedure be used to generate task environment models for task planning? Possibly, but there is no strict “query” to answer except for the following: “how should the environment be modelled in order to maximize performance?”. Levy et al. used model fragments of different detail level which was structured in so called *assumption classes* which represent the partial order of complexity and detail between the possible model fragments. The same idea can be used to generate task environment models for planning but another method than the one described in [43] will probably be used.

Bibliography

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.

[2] J. Aldrich. R. A. Fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12:162–176, 1997.

[3] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

[4] P. Athanasios and P. S. Unnikrishna. *Probability, Random Variables and Stochastic Processes*. Mcgraw-Hill Education, 2002.

[5] F. Bacchus and R. Petrick. Modeling an agent's incomplete knowledge during planning and execution. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 432–443, 1998.

[6] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[7] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Verlag, 1980.

[8] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a model based planner. In *IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[9] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press Inc., New York, 1995.

[10] R. J. Brachman and H. Levesque. *Knowledge Representation an Reasoning*. Morgan Kaufmann, 2004.

[11] G. Conte. Navigation functionalities for an autonomous uav helicopter. *Licentiate Thesis Linköping Institute of Technology at Linköping University*, 2007.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2000.

[13] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Artificial Intelligence*, 93(1-2):1–27, 1989.

[14] T. Dean and M. Welman. *Planning and Control.* Morgan Kaufmann, 1991.

[15] T. L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.

[16] T. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.

[17] P. Doherty. Advanced research with autonomous unmanned aerial vehicles. *Proceedings on the 9th International Conference on Principles of Knowledge Representation and Reasoning*, 2004.

[18] P. Doherty, P. Haslum, F. Heintz, T. Merz, P. Nyblom, T. Persson, and B. Wingman. A distributed architecture for autonomous unmanned aerial vehicle experimentation. *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2004.

[19] P. Doherty and F. Heintz. A knowledge processing middleware framework and its relation to the JDL data fusion model. *Journal of Intelligent and Fuzzy Systems*, 17(4), 2006.

[20] R. Durrett. *Probability: Theory and Examples.* Duxbury press, 2004.

[21] K. Erol, J. Hendler, and D. Nau. Semantics for hierarchical task-network planning. Technical report, University of Maryland Institute for Advanced Computer Studies, 1994.

[22] B. Falkenhainer and K. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[23] R. J. Firby. *Adaptive Execution in Complex Dynamic Domains.* PhD thesis, Yale University, 1989.

[24] R. Fourer, M. Gay, and W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* The Scientific Press, 1993.

[25] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *AAAI/IAAI*, pages 343–349, 1999.

[26] M. Fox and D. Long. An extension of PDDL for expressing temporal planning domains. *Journal of AI Research*, 20:61–124, 2003.

[27] M. Freed. *Simulating Human Performance in Complex, Dynamic Environments.* PhD thesis, Northwestern University, 1998.

[28] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan. Towards an application framework for automated planning and scheduling. In *Proceedings of the IEEE Aerospace Conference*, 1997.

[29] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia, Italy, 2005.

[30] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—the planning domain definition language. Technical report, AIPS-98 Planning Committee, 1998.

[31] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning, theory and practice.* Morgan Kaufmann Publishers, 2004.

[32] G. Granlund, K. Nordberg, J. Wiklund, P. Doherty, E. Skarman, and E. Sandewall. An intelligent autonomous aircraft using active vision. In *Proceedings of the UAV 2000 International Technical Conference and Exhibition*, 2000.

[33] J. A. Hartigan. *Clustering algorithms.* New York: John Wiley, 1975.

[34] P. Haslum. Prediction as a knowledge representation problem: A case study in model design. *Licentiate Thesis Linköping Institute of Technology at Linköping University.*, 2002.

[35] B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ, 2002.

[36] D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms.

[37] A. Jonsson and A. G. Barto. Automated state abstraction for options using the u-tree algorithm. In *Advances in Neural Information Processing Systems*, pages 1054–1060, 2000.

[38] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[39] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[40] C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

[41] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[42] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[43] Alon Y. Levy, Yumi Iwasaki, and Richard Fikes. Automated model selection for simulation based on relevance reasoning. *Artificial Intelligence*, 96(2):351–394, 1997.

[44] L. Ljung and T. Glad. *Modellbygge och Simulering*. Studentlitteratur, 2004.

[45] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the 21st International Conference on Machine Learning*.

[46] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

[47] A. Meystel. Knowledge based nested hierarchical control. *Advances in Automation and Robotics*, 2:63–152, 1990.

[48] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[49] K. Murphy. A survey of POMDP solution techniques. http://www.cs.ubs.ca/ murphy/Papers/pomdp.pdf, 2005.

[50] R. R. Murphy. *Introduction to AI Robotics*. The MIT Press, 2000.

[51] N. Muscettola. HSTS: Integrating planning and scheduling. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[52] K. L. Myers. A procedural knowledge approach to task-level control. In *Proceedings of the Third International Conference on AI Planning Systems*, 1996.

[53] D. Nau, T. C. Au, O. Ilghami O., U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[54] N. J. Nilsson. Shakey the robot. Technical report, AI Center, SRI International, 1984.

[55] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer Verlag, 2006.

[56] P. Nyblom. A language translator for robotic task procedure specifications. Master's thesis, Linköping Univirsity.

[57] P. Nyblom. Dynamic abstraction for hierarchical problem solving and execution in stochastic dynamic environments. In *Starting AI Researcher Symposium (STAIRS)*, 2006.

[58] P. Nyblom. Dynamic problem generation in a UAV domain. In *The 6th IFAC Symposium on Intelligent Autonomous Vehicles*, 2007.

[59] Object Management Group (OMG). *Common Object Request Broker Architecture: Core Specification*, 2004.

[60] R. Parr. *Hierarchical Control and Learning for Markov Decision Processes.* PhD thesis, University of California at Berkley, 1998.

[61] P. O. Pettersson. Sampling-based path planning for an autonomous helicopter. *Licentiate Thesis Linköping Institute of Technology at Linköping University.*, 2006.

[62] M. Puterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming.* Wiley Inter-science, 1994.

[63] S. Russel and P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 2003.

[64] E. D. Sacerdoti. Planning in a hiearchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[65] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and sons, 1998.

[66] K. Steinkraus and L. P. Kaelbling. Combining dynamic abstractions in large mdps, 2004.

[67] N. Sturtevant and M. Buro. Partial pathfinding using map abstraction and refinement. In *Proceedings of the AAAI National Conference on Artificial Intelligence.* AAAI, 2005.

[68] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, 1990.

[69] R. S. Sutton and A. G. Barto. *Reinforcement Learning An Introduction.* The MIT Press, 1998.

[70] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[71] A. Tate. Project planning using a hierarchic non-linear planner. Technical report, Department of Artificial Intelligence, University, 1975.

[72] A. Tate, B. Drabble, and J. Dalton. O-plan: a knowledge-based planner and its application to logistics. *Advanced Planning Technology, AAAI Press*, 1996.

[73] G. Theocharous, S. Mahadevan, and L. P. Kaelbling. Spatial and temporal abstractions in pomdps applied to robot navigation, 2005.

[74] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics.* 2005.

[75] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers Inc., San Francisco, CA, 1988.

[76] D. E. Wilkins and K. L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6):731–761, 1995.

[77] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227, 1995.

Avdelning, Institution
Division, Department

AIICS,
Dept. of Computer and Information Science
581 83 Linköping

Datum
Date

April 24 2008

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ ______________

Rapporttyp
Report category

☒ Licentiatavhandling
☐ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ ______________

ISBN
978-91-7393-905-8

ISRN
LiU-Tek-Lic–2008:21

Serietitel och serienummer
Title of series, numbering

ISSN
0280–7971

Linköping Studies in Science and Technology

Thesis No. 1363

URL för elektronisk version

Titel
Title

Dynamic Abstraction for Interleaved Task Planning and Execution

Författare
Author

Per Nyblom

Sammanfattning
Abstract

It is often beneficial for an autonomous agent that operates in a complex environment to make use of different types of mathematical models to keep track of unobservable parts of the world or to perform prediction, planning and other types of reasoning. Since a model is always a simplification of something else, there always exists a tradeoff between the model's accuracy and feasibility when it is used within a certain application due to the limited available computational resources. Currently, this tradeoff is to a large extent balanced by humans for model construction in general and for autonomous agents in particular. This thesis investigates different solutions where such agents are more responsible for balancing the tradeoff for models themselves in the context of interleaved task planning and plan execution. The necessary components for an autonomous agent that performs its abstractions and constructs planning models dynamically during task planning and execution are investigated and a method called DARE is developed that is a template for handling the possible situations that can occur such as the rise of unsuitable abstractions and need for dynamic construction of abstraction levels. Implementations of DARE are presented in two case studies where both a fully and partially observable stochastic domain are used, motivated by research with Unmanned Aircraft Systems. The case studies also demonstrate possible ways to perform dynamic abstraction and problem model construction in practice.

Nyckelord
Keywords Dynamic Abstraction, Task Planning, Automatic Model Construction

Department of Computer and Information Science
Linköpings universitet

Linköping Studies in Science and Technology
Faculty of Arts and Sciences - Licentiate Theses

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)
No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.
No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.
No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.
No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.
No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.
No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.
No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.
No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.
No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.
No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.
No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.
No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.
No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.
No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.
No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.
No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.
No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.
No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.
No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.
No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.
No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.
No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.
No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.
No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.
No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.
No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.
No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.
No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.
No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.
No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.
No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.
No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.
No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.
No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.
No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.
No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.
No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.
No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge-Bases, 1991.
No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.
No 318 **Lena Srömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.
No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.
No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.
No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.
No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.
No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Sytems, 1992.
No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.
No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.
No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.
No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.
No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.
No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.
No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.

No 402	**Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406	**Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414	**Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agent-teoretiskt perspektiv, 1994.
No 417	**Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436	**Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437	**Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440	**Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94	**Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94	**Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441	**Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446	**Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450	**Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451	**Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452	**Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455	**Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94	**Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462	**Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463	**Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464	**Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469	**Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473	**Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475	**Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476	**Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478	**Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95	**Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylkprogram, 1995.
No 482	**Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488	**Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489	**Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497	**Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498	**Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503	**Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95	**Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95	**Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513	**Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517	**Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518	**Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522	**Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538	**Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545	**Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546	**Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96	**Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549	**Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550	**Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557	**Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558	**Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561	**Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563	**Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567	**Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.
No 575	**Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.
No 576	**Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.
No 587	**Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.
No 589	**Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591	**Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.
No 595	**Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.
No 597	**Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598	**Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.
No 599	**Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.
No 607	**Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.
No 609	**Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.
FiF-a 4	**Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.
FiF-a 6	**Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.
No 615	**Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.
No 623	**Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.
No 626	**David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.
No 627	**Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.
No 629	**Gunilla Ivefors:** Krigsspel coh Informationsteknik inför en oförutsägbar framtid, 1997**.**
No 631	**Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997
No 639	**Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.
No 640	**Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.
No 643	**Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.
No 653	**Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.
FiF-a 13	**Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.
No 674	**Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.
No 676	**Jan Håkegård**: Hiera rchical Test Architecture and Board-Level Test Controller Synthesis, 1998.
No 668	**Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
No 675	**Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.
FiF-a 14	**Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.
No 695	**Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.
No 700	**Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.
FiF-a 16	**Marie-Therese Christiansson:** Inter-organistorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 712	**Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.
No 719	**Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.
No 723	**Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.
No 725	**Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.
No 730	**Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.
No 731	**Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.
No 733	**Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.
No 734	**Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.
FiF-a 21	**Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.
FiF-a 22	**Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.
No 737	**Jonas Mellin:** Predictable Event Monitoring, 1998.
No 738	**Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.
FiF-a 25	**Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.
No 742	**Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.
No 748	**Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.
No 751	**Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader,1999.
No 752	**Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.
No 753	**Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.
No 754	**Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.
No 766	**Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769	**Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775	**Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30	**Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787	**Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 788	**Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
No 790	**Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791	**Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800	**Anders Subotic:** Software Quality Inspection, 1999.
No 807	**Svein Bergum**: Managerial communication in telework, 2000.

No 809	**Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32	**Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808	**Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820	**Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
No 823	**Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
No 832	**Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
FiF-a 34	**Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
No 842	**Magnus Kald:** The role of management control systems in strategic business units, 2000.
No 844	**Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
FiF-a 37	**Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.
FiF-a 40	**Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
FiF-a 41	**Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
No. 854	**Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
No 863	**Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
No 881	**Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
No 882	**Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
No 890	**Annika Flycht-Eriksson:** Domain Knowledge Management inInformation-providing Dialogue systems, 2001.
FiF-a 47	**Per-Arne Segerkvis**t: Webbaserade imaginära organisationers samverkansformer: Informationssystemarkitektur och aktörssamverkan som förutsättningar för affärsprocesser, 2001.
No 894	**Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
No 906	**Lin Han**: Secure and Scalable E-Service Software Delivery, 2001.
No 917	**Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
No 916	**Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
FiF-a-49	**Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
FiF-a-51	**Per Oscarsson:**Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
No 919	**Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
No 915	**Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
No 931	**Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
No 933	**Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
No 938	**Bourhane Kadmiry**: Fuzzy Control of Unmanned Helicopter, 2002.
No 942	**Patrik Haslum**: Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
No 956	**Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
FiF-a 58	**Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
No 964	**Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
No 973	**Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.
No 958	**Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.
FiF-a 61	**Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.
No 985	**Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.
No 982	**Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.
No 989	**Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.
No 990	**Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.
No 991	**Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.
No 999	**Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002
No 1000	**Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.
No 1001	**Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
No 988	**Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
FiF-a 62	**Lennart Ljung:** Utveckling av en projektivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
No 1003	**Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
No 1005	**Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
No 1008	**Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
No 1010	**Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
No 1015	**Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
No 1018	**Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
No 1022	**Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
FiF-a 65	**Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.

No 1024	**Aleksandra Tešanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.
No 1034	**Arja Vainio-Larsson**: Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
No 1033	**Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befarade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
FiF-a 69	**Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
No 1049	**Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
No 1052	**Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
No 1054	**Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
FiF-a 71	**Emma Eliason:** Effektanalys av IT-systems handlingsutrymme, 2003.
No 1055	**Carl Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
No 1058	**Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
FiF-a 73	**Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
No 1079	**Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
No 1084	**Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
FiF-a 74	**Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
No 1094	**Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
No 1095	**Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, 2004.
No 1099	**Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
No 1110	**Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
No 1116	**Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
FiF-a 77	**Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
No 1126	**Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
No 1127	**Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
No 1132	**Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
No 1130	**Ioan Chisalita:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
No 1138	**Thomas Gustafsson:** Maintaining Data Consistency im Embedded Databases for Vehicular Systems, 2004.
No 1149	**Vaida Jakoniené:** A Study in Integrating Multiple Biological Data Sources, 2005.
No 1156	**Abdil Rashid Mohamed**: High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
No 1162	**Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
No 1165	**Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
FiF-a 84	**Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
No 1166	**Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
No 1167	**Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
No 1168	**Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
FiF-a 85	**Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
No 1171	**Yu-Hsing Huang:** A systemic traffic accident model, 2005.
FiF-a 86	**Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
No 1172	**Petter Ahlström**: Affärsstrategier för seniorbostadsmarknaden, 2005.
No 1183	**Mathias Cöster**: Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
No 1184	**Åsa Horzella**: Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
No 1185	**Maria Kollberg**: Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.
No 1190	**David Dinka**: Role and Identity - Experience of technology in professional settings, 2005.
No 1191	**Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.
No 1192	**Nicklas Bergfeldt:** Towards Detached Communication for Robot Cooperation, 2005.
No 1194	**Dennis Maciuszek:** Towards Dependable Virtual Companions for Later Life, 2005.
No 1204	**Beatrice Alenljung:** Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005
No 1206	**Anders Larsson:** System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.
No 1207	**John Wilander:** Policy and Implementation Assurance for Software Security, 2005.
No 1209	**Andreas Käll:** Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.
No 1225	**He Tan:** Aligning and Merging Biomedical Ontologies, 2006.
No 1228	**Artur Wilk:** Descriptive Types for XML Query Language Xcerpt, 2006.
No 1229	**Per Olof Pettersson:** Sampling-based Path Planning for an Autonomous Helicopter, 2006.
No 1231	**Kalle Burbeck:** Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.
No 1233	**Daniela Mihailescu:** Implementation Methodology in Action: A Study of an Enterprise Systems Implementation Methodology, 2006.
No 1244	**Jörgen Skågeby:** Public and Non-public gifting on the Internet, 2006.
No 1248	**Karolina Eliasson:** The Use of Case-Based Reasoning in a Human-Robot Dialog System, 2006.
No 1263	**Misook Park-Westman:** Managing Competence Development Programs in a Cross-Cultural Organisation-What are the Barriers and Enablers, 2006.
FiF-a 90	**Amra Halilovic:** Ett praktikperspektiv på hantering av mjukvarukomponenter, 2006.
No 1272	**Raquel Flodström**: A Framework for the Strategic Management of Information Technology, 2006.

No 1277	**Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Embedded Systems, 2006.
No 1283	**Håkan Hasewinkel:** A Blueprint for Using Commercial Games off the Shelf in Defence Training, Education and Research Simulations, 2006.
FiF-a 91	**Hanna Broberg:** Verksamhetsanpassade IT-stöd - Designteori och metod, 2006.
No 1286	**Robert Kaminski:** Towards an XML Document Restructuring Framework, 2006
No 1293	**Jiri Trnka:** Prerequisites for data sharing in emergency management, 2007.
No 1302	**Björn Hägglund:** A Framework for Designing Constraint Stores, 2007.
No 1303	**Daniel Andreasson**: Slack-Time Aware Dynamic Routing Schemes for On-Chip Networks, 2007.
No 1305	**Magnus Ingmarsson:** Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, 2007.
No 1306	**Gustaf Svedjemo**: Ontology as Conceptual Schema when Modelling Historical Maps for Database Storage, 2007.
No 1307	**Gianpaolo Conte**: Navigation Functionalities for an Autonomous UAV Helicopter, 2007.
No 1309	**Ola Leifler**: User-Centric Critiquing in Command and Control: The DKExpert and ComPlan Approaches, 2007.
No 1312	**Henrik Svensson**: Embodied simulation as off-line representation, 2007.
No 1313	**Zhiyuan He:** System-on-Chip Test Scheduling with Defect-Probability and Temperature Considerations, 2007.
No 1317	**Jonas Elmqvist:** Components, Safety Interfaces and Compositional Analysis, 2007.
No 1320	**Håkan Sundblad:** Question Classification in Question Answering Systems, 2007.
No 1323	**Magnus Lundqvist**: Information Demand and Use: Improving Information Flow within Small-scale Business Contexts, 2007.
No 1329	**Martin Magnusson:** Deductive Planning and Composite Actions in Temporal Action Logic, 2007.
No 1331	**Mikael Asplund:** Restoring Consistency after Network Partitions, 2007.
No 1332	**Martin Fransson:** Towards Individualized Drug Dosage - General Methods and Case Studies, 2007.
No 1333	**Karin Camara:** A Visual Query Language Served by a Multi-sensor Environment, 2007.
No 1337	**David Broman:** Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments, 2007.
No 1339	**Mikhail Chalabine:** Invasive Interactive Parallelization, 2007.
No 1351	**Susanna Nilsson:** A Holistic Approach to Usability Evaluations of Mixed Reality Systems, 2008.
No 1353	**Shanai Ardi:** A Model and Implementation of a Security Plug-in for the Software Life Cycle, 2008.
No 1356	**Erik Kuiper:** Mobility and Routing in a Delay-tolerant Network of Unmanned Aerial Vehicles, 2008.
No 1359	**Jana Rambusch**: Situated Play, 2008.
No 1363	**Per Nyblom:** Dynamic Abstraction for Interleaved Task Planning and Execution, 2008.