Machine Learning and Adaptation of Domain Models to Support Real Time Planning in Autonomous Systems

Machine Learning and Adaptation of Domain Models to Support Real Time Planning in Autonomous Systems

Month 6 Report

Austin Tate, Gerhard Wickler
University of Edinburgh

Lee McCluskey, Lukáš Chrpa
University of Huddersfield

Contents

Chapter 1

Introduction

The research hypothesis of this project is as follows:

> Automatically learning and adapting an accurate and adequate domain model for the purposes of symbolic reasoning, in particular for the processes of automated planning, enables effective, sustained goal-directed behavior for real time dynamic autonomous systems.

This is a working document which is written to encapsulate much of the fundamentals of this research project, and existing work underlying the research hypothesis. It forms deliverable "D2" in the original proposal. It includes background material on planning algorithms, domain model languages and learning methods.

Chapter 2

Automated Planning

Automated planning [Ghallab *et al.*, 2004] deals with a problem of finding sequences of actions transforming the environment from some initial state to a desired goal state. In other words, automated planning is about reasoning how autonomous entities are going to act to achieve their goals. Despite belonging to a class of computationally very hard problems, automated planning has a wide range of practical applications, especially where applications of autonomous agents or robots are necessary. In the space exploration, one of the first successful applications of planning was the Deep Space 1 mission [Bernard *et al.*, 2000]. The mission featured the application of the Autonomous Remote Agent system [Muscettola *et al.*, 1998], software based on planning techniques. Besides the space exploration there are many other more mundane applications of planning, for example planning of manufacturing processes [Nau *et al.*, 1995] or planning autonomous vehicles in rescue missions [Dave *et al.*, 2003; Teichteil-Königsbuch and Fabiani, 2006].

Automated planning can be divided into several categories which differs by levels of expressiveness (as discussed below). Straightforwardly, there is a correlation between the level of expressiveness and computational complexity in terms of more expressiveness implies higher complexity. Real-world problems usually require a high level of expressiveness (e.g. time, consumable resources, uncertainty, partial observability of the environment etc.) to encapsulate problems' properties in the most possible realistic way. Hence, the question is how to find a reasonable compromise, i.e., having realistic models of the problems which are also possible to solve.

The prevailing view in the planning community is that planning problems should be represented in some language (e.g. PDDL) which serves as a 'communication protocol' with generic planning engines (planning problem solvers). A level of expressiveness will give insights into which language

and planning engine(s) can be applied for modeling and solving (real-world) planning tasks. An inspiration can be taken from tools for modeling planning tasks (e.g. GIPO), learning planning operator schema (e.g. LOCM) or learning structural knowledge of planning tasks which can be used for planning problem reformulation (e.g. macro-operators). We will consider such systems in more detail later in the document.

For now we make the basic assumption that planning engines (programs that do planning):

> input a planning task which consists of 2 inputs: a *domain model* describing the area of planning application (spacecraft manoeuvres, lift scheduling, robot manipulation etc) in terms of possible actions (represented as operators, object classes etc), and a *problem description* which is the particular planning task to be solved (represented as an initial state and a goal or task statement, etc).

2.1 Categories of Planning

This section describes main categories of automated planning which will be considered in the project.

2.1.1 Classical Planning

Classical planning (in state space) deals with finding a sequence of actions transforming the static, deterministic and fully observable domain model from some initial state to a desired goal state [Ghallab *et al.*, 2004].

Features of a domain are described by *atoms* which are either propositions or predicates. For example, a situation where a block b is stacked on a block a can be represented by a predicate `on(b,a)`. *States* are defined as sets of (ground) atoms. A *planning operator* $o = (name(o), pre(o), eff^-(o), eff^+(o))$ is a construct, where $name(o) = op\_name(x_1, \ldots, x_k)$ (op_name is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator and $pre(o)$ (a precondition of o) ,$eff^-(o)$ (negative effects of o) and $eff^+(o)$ (positive effects of o) are sets of atoms. An action a is *applicable* is a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus eff^-(a)) \cup eff^+(a)$. For example, a planning operator for unstacking blocks can be defined as follows: o = (*name*(*o*)=`unstack(?x,?y)`, *pre*(*o*)=`{on(?x,?y),clear(?x),handempty}`, $eff^-(o)$=`{on(?x,?y),clear(?x),handempty}`, $eff^+(o)$) = {`holding(?x)`, `clear(?y)`}). In plain words, it says that we can unstack a block `?x` from a

block `?y` if `?x` is stacked on `?y`, `?x` is clear (i.e., nothing is stacked on it) and the robotic hand is empty (i.e., it does not hold any block). Unstacking a block `?x` from a block `?y` results in a state where `?x` is no longer stacked on `?y`, the robotic hand is no longer empty but holds `?x` and as a consequence `?y` becomes clear because no block is stacked on it (`?x` is no longer there). *Actions* are instances of planning operators, i.e., they can be obtained by grounding (substituting constants for variables).

A *domain model* is specified via sets of predicates and planning operators (alternatively propositions and actions). A **planning task** is specified via a planning domain model, a set of objects (constants), an initial state and set of goal atoms. A *plan* is a sequence of actions. A plan is a *solution* of a planning problem if and only if a consecutive application of the actions in the plan (starting in the initial state) results in a state, where all the goal atoms are satisfied.

Despite being the easiest form of automated planning the computational complexity of classical planning is up to PSPACE-complete [Bylander, 1994]. On the other hand, some classes of classical planning problems can be computationally easy (in P) [Helmert, 2003]. Thanks to the International Planning Competition (IPC)[1] many advanced planning engines have been developed. Well known and successful planning engines include FF [Hoffmann and Nebel, 2001], LPG [Gerevini *et al.*, 2004], SATPLAN [Kautz *et al.*, 2006] and LAMA [Richter and Westphal, 2010].

2.1.2 Temporal Planning

Temporal planning extends classical planning by considering time explicitly. Actions (or planning operators) do not have immediate effects but their execution takes some (pre-defined) time. Those actions are called durative actions. A definition of a planning operator (or an action) is extended as follows. $dur(o)$ is a duration of execution of an operator o and is represented by a non-negative real number or integer. Positive and negative effects are split into four sets (two sets each) representing effects taking place at time the operator is executed and effects taking place at the time that execution of the operator finishes, i.e., $eff^-_{start}(o)$, $eff^+_{start}(o)$, $eff^-_{end}(o)$, $eff^+_{end}(o)$. For example, an operator o such that $name(o)$=`move(?r,?l1,?l2)` which moves a robot `?r` from a location `?l1` to a location `?l2` can be represented in temporal planning as follows. $dur(o)$=1000s, $pre(o)$ = {`at(?r,?l1)`,`path(?l1,?l2}`, $eff^-_{start}(o)$ = {`at(?r,?l1)`}, $eff^+_{start}(o)$ = {`onway(?r)`}, $eff^-_{end}(o)$ = {`onway(?r)`} and $eff^+_{end}(o)$ = {`at(?r,?l2)`}. In plain words it says that

[1] http://ipc.icaps-conference.org

a robot `?r` when started moving is no longer at `?l1` but on the way until it reaches (after 1000s) `?l2`. Contrary to this, in classical planning it is not necessary to consider when the robot is on the way because the robot 'teleports' itself between locations instantly.

Temporal planning is often applied together with scheduling because temporal planners provide (partial) ordering of actions (in scheduling also called activities) while schedulers place these actions (activities) into time windows.

Recent implementations of temporal planners include LPG [Gerevini *et al.*, 2004], Crickey [Coles *et al.*, 2008] or Filuta [Dvořák and Barták, 2010].

2.1.3 Conformant Planning

Conformant planning extends classical planning by considering uncertainty in terms of partial observability of the environment and/or non-deterministic effects of planning operators (or actions). Its goal is to attempt to find a plan that will work under any of the possible effects or actual configurations of the initial state (this is a *conformant plan*). Atoms besides being *true* (present in a state) or *false* (not present in a state) can be also *unknown*. If an atom is unknown then there is no evidence (due to incomplete information) whether the atom is true or false in a given state. Applicability of an action a (or a planning operator) in a state s is then defined in such a way that $\forall p \in pre(a)$ p is true in s. For example, we might not know whether there is a path between certain locations $l1$ and $l2$. Hence, we cannot execute an action `move(r,l1,l2)` until we somehow reveal the existence of the path.

Positive and negative effects of actions can be split into several sets representing different alternatives of outcomes of execution of these actions, i.e., $eff_1^-(o), eff_1^+(o), \ldots, eff_k^-(o), eff_k^+(o)$. Probabilities of occurrences of particular alternatives may be pre-defined. For example, the `unstack(?x,?y)` might result in a state where either the robotic hand holds `?x`, `?x` has fallen to the table or `?y` has fallen to the table etc.

Existing conformant planner are, for instance, CPA [Tran *et al.*, 2008], Gamer [Kissmann and Edelkamp, 2008] or $\langle K, K_0 \rangle$-planner [Albore *et al.*, 2010].

2.1.4 Continuous Planning

Continuous Planning differs from the previous categories of automated planning in that continuous changes of object values over time are expressed and reasoned with by the planner. Thus the "Continuous" in the title relates to the kind of expressions in the domain model, and not the kind of planning (i.e. it does *not* mean *continuously planning*). Basically, the environment

is represented by a set of *object fluents* rather than a set of propositions (or grounded predicates). These fluents are affected by (running) *processes* represented by functions of time. Actions are, in this case, used for executing or stopping processes. Besides actions there are *events* which are also used for executing or stopping processes. Events are triggered or may trigger automatically when certain conditions are met. For example, an action `move(r,l1,l2)` triggers a process `moving(r,l1,l2)` which updates the position of the robot r with respect to an actual distance it traveled from $l1$ to $l2$. An event `stopat(r,l2)` is eventually triggered after r reaches $l2$.

Continuous Planning can model real-world problems in a very realistic way, however, continuous planning problems might be undecidable. Complexity can be reduced by discretizing time and object values (fluents) which could still keep the model realistic. Continuous planning is quite a new research field, although a stable version of PDDL exists with which to express such domains (PDDL+ [Fox and Long, 2006]).

2.1.5 Hierarchical Task Networks

All the approaches described so far have the following in common: they consider plans as homogeneous ("flat") sets of actions assembled to achieve a goal. That is, none of the actions in a plan is considered more abstract than another. On the contrary, *hierarchical* approaches to planning have appeared as early as the mid-70s [Sacerdoti, 1974; Tate, 1977]. For this approach to be most effective, rather than specifying the problem using a goal to be achieved, we use a task to be carried out. So in a "task based" scenario, the *achieve goal* of "thirst quenched" would rather be posed as the abstract *task* "consume some water". For a long time, this type of planner has been used to implement practical planning applications as other approaches lacked necessary performance and did not correspond to expert knowledge in practical domains.

In Hierarchical Task Network (HTN) planning, a planing domain is described by a set of operators as defined above. In addition, a set of methods is defined, where a method consists of a parameterized name (like an operator name), a task that can be accomplished with the method, a network of sub-tasks, and a set of constraints. The exact definitions of these components depend on the planner used and there is no standard.

For example, the task accomplished by a method can be described by pattern consisting of a task name and some variables describing the parameters of the task. A task could be to shift a stack of blocks from one location to another: `(shift-stack ?l1 ?l2)`, where `?l1` and `?l2` represent the source and target locations. The sub-tasks for this method could be

two tasks, namely shifting the topmost block and shifting the remainder of the stack: `(t1:shift-block ?l1 ?l2)` and `(t2: shift-stack ?l1 ?l2)`. Constraints on this method enforce the order between the two subtasks (`t1` $\langle$ `t2`) and the precondition that there is at least one block at the source location: `(not (empty ?l1))`. Another method to shift the empty stack would complete this domain.

An important point here is the power of the formalism: different preconditions can render alternative methods applicable, which corresponds to branching constructs in traditional programming languages; and methods may have as sub-tasks the same type of task as the overall method accomplishes, which corresponds to recursion in traditional programming languages. Hence, HTN planning problems are in general undecidable.

Planners that handle HTN planning problems are often more concerned with rich sets of constraints and contain features that allow direct mapping of parts of the representation to code fragments, which obscures theoretic concerns like soundness and completeness are not an issue.

The SIPE planner [Wilkins, 1988] and the O-Plan system [Currie and Tate, 1991] fall into the category of HTN planners that have been used in many practical application domains. A planner that has a more solid theoretical foundation is the SHOP system described in [Nau *et al.*, 2001].

2.1.6 Knowledge-Rich Representations

In the 1990s the initiative in Knowledge-Sharing inspired a new strand of research into knowledge-rich plan representations that could be used to communicate plans between agents. The aim here was to have a maximum of expressivity, though no planners were developed specifically to output these representations.

The prime motivation for the development of the Core Plan Representation (CPR) [Pease and Carrico, 1996] was to address the plan interchange requirements of several military planning systems. Just like there are a number of different planning problems and approaches in AI, there are different representations that come with the different systems that implement the various approaches. Furthermore, systems that process plans, e.g. workflow systems, control systems, etc., will need to exchange information with the AI plan-generating systems. And we must not forget the human in the loop: people need to understand plans in order to execute them in line with the intent that underlies them.

The result of the CPR effort is a small ontology that defines a few core concepts that are expected to be shared between most systems that deal with plans. The way these concepts are defined relies on ontological principles:

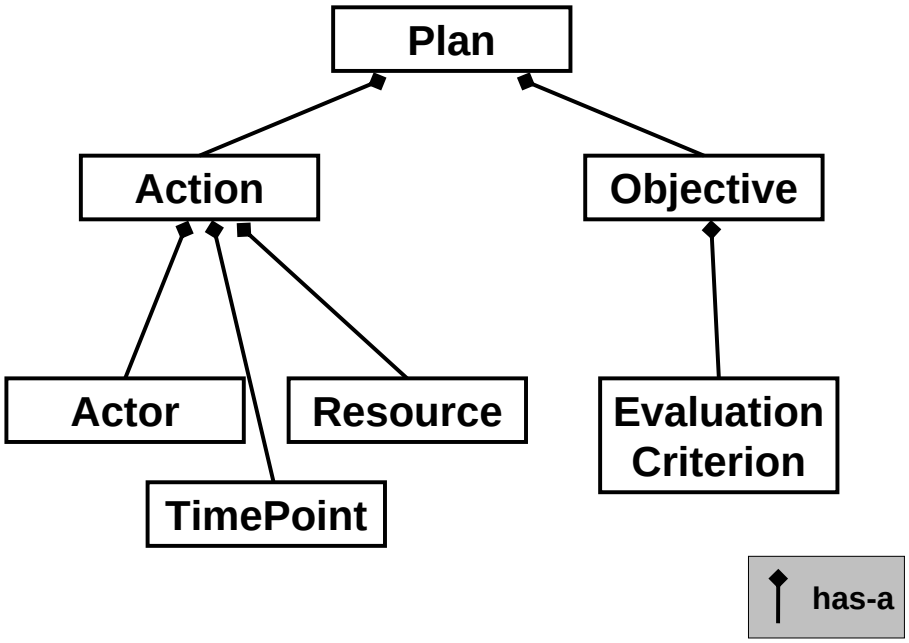

Figure 2.1: Main concepts of the CPR

each concept is defined by the relations that must hold with other concepts and some internal structure. Furthermore, there is human-readable text that explains the concepts, which is not in a form processable by any planning system.

The main concepts that form the CPR are shown in figure 2.1. The most important relation that holds between these concepts is the "has a" relation that indicates that an instance of one concept has a component that is an instance of the other concept. Alternatively, the second concept forms a building block for the former. From an ontological point of view this is slightly unusual as, in most ontologies, the "is a" relation is the most prominent relation.

The CPR was developed further in various projects, resulting in more comprehensive ontologies and representations [Tate, 1998].

2.2 Representing Domain Knowledge in Planning

2.2.1 Basic Definitions

A key part of any Automated Planning activity is to do with "domain modeling", that is creating a *formal model* of the domain (reality) that the planning is to occur. Having introduced the idea above, we will discuss this in more detail, as modeling, and its automation, is a key part of the project.

Assume X is a formal model of Y, and for example assume Y is some physical domain. Then:

- symbols in X are used to represent features in Y. A symbol in X might be "above", which represents the spatial relation in the physical domain Y.

- statements in X are used to represent what is true in the domain. So we might state "block a is above block b" in X if that is true in the physical world.

- action representations in X (often called operators) represent actual actions in Y. Simulating the operation of some action "put b on a" in X should make the statement "above b on a" true, as it does in the physical world.

Clearly, good models conform to the reality that they are modeling. Simulating the operation of an action in the model should give the results we expect to see in the real world. The model is "formal" if it has a precise syntax (which means it can be parsed by a program), and it has precise operational semantics (which means, amongst other things, that a program can perform the simulation of the action).

A great deal of effort is needed to precisely model real domains for planning. For example, it is well known that the domain models used in the AI Planning operations used at NASA required a team of highly specialized knowledge engineers to do the encoding. To consider this area in more depth, we make some definitions about quality factors of a domain model, with respect to a set of planning task requirements. A domain model is:

- *accurate* if the features it contains conform to the requirements. For example, relationships depicted in the model are deemed true in the requirements, and the effect of actions in the environment represent the real effects of actions faithfully.

- *adequate* if it represents in sufficient detail the planning task. For example, we may say that a representation of a domain where all actions execute instantaneously is inadequate, if there is a need in the requirements to represent durative actions. .

- *complete* if it is adequate and accurate, and it contains sufficient features to satisfy the requirements.

Accuracy is therefore related to correctness, but while the latter quality is generally considered a relationship between two formal expressions, accuracy relates a formal expression (the domain model) with an informal statement (the requirements). Domain models can be adequate but not accurate (they represent the task at the required level of abstraction, but some of the features are not represented faithfully) or accurate but not adequate (all the features present conform to the requirements, but some requirements cannot be represented at all). A model can be accurate and adequate, but not complete: this is the case where the model does not contain all the features required - for example some parts of the requirements may be missing.

These terms are used in a very similar manner in Software Engineering, when one is interested in creating a formal requirements model out of a set of informal requirements [**?**].

2.2.2 Examples of Existing Representation Languages

This section describes existing languages used for modeling planning domains associated tasks.

PDDL

Planning Domain Definition Language (PDDL) [Ghallab *et al.*, 1998] is the best-known language which is used for modeling domains and planning problems. PDDL is a LISP-like language[2] and thanks to the IPC PDDL became very popular in the planning community and majority of the planning engines, therefore, use PDDL. There are three significant milestones in the development of PDDL.

PDDL 1.2 — This version was introduced on the 1st IPC in 1998. It separates planning task description into domain models and planning problem description. It is suitable for classical planning and incorporates some additional features such as object types (used in grounding) or conditional effects.

[2]LISP is a well-known functional programming language

PDDL 2.1 — This version was introduced on the 3rd IPC in 2002. It extends the previous version by introducing numeric fluents (used for representing, for instance, fuel-level, energy, distance etc.) and durative actions. It is therefore suitable for temporal planning.

PDDL 3.0 — This version was introduced on the 5th IPC in 2006. It introduces hard constraints in a form of logical expressions, which must be true during the execution of the plan, soft constraints (preferences) also in a form of logical expressions, which increases quality of plans if true during the execution of the plan.

PDDL 3.1 — This version was introduced on the 6th IPC in 2008 and is the latest. It introduces object fluents whose range in not only numerical but could be any object type.

There are some extensions of PDDL. A well-known extension is Probabilistic PDDL (PPDDL) [Younes and Littman, 2004] which was introduced in probabilistic track of the 4th IPC in 2004. PPDDL supports actions with non-deterministic effects with probability distribution. However, PPDDL does not support patrial observability, therefore, it covers only a part of conformant planning. Another extension of PDDL is PDDL+ [Fox and Long, 2006], a language which allows to model continuous processes and events. Therefore, it might be suitable for modeling continuous planning tasks.

NDDL

New Domain Definition Language (NDDL) [Frank and Jónsson, 2003] has been developed by NASA in 2002. NDDL is a successor of HSTS and is primarily focused to space applications. NDDL mainly differs from PDDL by using state variable representation (object fluents) rather than propositional or predicate representation and activities and constraints between them rather than planning operators (actions). Each activity has a start and end time interval which encapsulates uncertainty of duration. Object fluents are affected by these activities and hence their values are developing through predefined timelines. The expressiveness of NDDL is able to model continuous planning tasks where time is discretized. NDDL seems to be more appropriate for modeling CAs, however, there is not a large portfolio of planning engines which supports NDDL (as it is for PDDL). The only planner which supports NDDL is NASA's EUROPA2 [Bernardini and Smith, 2007].

2.2.3 Domain Modeling: Manual Tools

Some of the earliest tools used for knowledge acquisition and modeling in AI planning came from the O-Plan [Tate *et al.*, 1994] and SIPE [David E. Wilkins, 1990] projects. O-Plan's Task Formalism and SIPE's Act Formalism were HTN domain model languages used in these projects, and both were written for dedicated planners. SIPE employed a graphical and textual editor, called the "Act Editor", to help users create domain models.

GIPO was one of the first general, planner and domain-independent tools appearing in the literature for supporting knowledge acquisition and modeling [Simpson, 2007]. The GIPO interface was designed to overcome syntax errors, and to make the task of creating a formal specification easier for a non-expert user. GIPO's GUI uses diagrams to support the definition of domain elements in a object-oriented approach. A significant advance on the original GUI interface was introduced in the third version of GPO[Simpson *et al.*, 2001], GIPO III[Simpson *et al.*, 2007]: the *Life History Editor* provides an interface where users represent the dynamics of an object class by constructing and annotating graphical state machines; the interface then *automatically* builds a formal specification of the domain in the form of a model in PDDL. As is often the case with methods where users construct diagrams within a GUI, however, once a generated domain model has been further edited outside of the GUI, the graphical construction cannot be regenerated, and any further maintenance has to be made using a manual method.

itSIMPLE provides tools and methods to support designers during domain model creation through an object-oriented approach [Vaquero *et al.*, 2007]. The user creates UML diagrams to represent aspects of the domain such as use case diagrams, and refines these into more detailed class diagrams, state machine diagrams, timing diagrams, and object diagrams etc. As in GIPO, the dynamics of operators are modeled using state machine diagrams to represent the states that a class object can go through during its lifetime. UML's constraint language (OCL) is used to capture an actions's pre- and post-conditions. Both GIPO and itSIMPLE generate PDDL as output, and have been used widely for creating models.

EUROPA [Barreiro *et al.*, 2012] is NASA's GUI for encoding domain models in NDDL. Like GIPO and itSIMPLE, it follows an object-oriented modeling approach, but focusses on a timeline-based representation of objects as discussed above in the section describing NDDL. Recently a higher level language ANML has been introduced in the EUROPA framework, with an associated editor, so that one can visualize the object type hierarchy and the relationships between actions, fluents, and objects. In particular, using the ANML editor, the engineer can inspect and analyze the fluent timelines, and

automatically translate the higher level model into solver-ready NDDL.

2.2.4 Domain Modeling: Automated Tools and Techniques

The key element of this research project in *automatically learning domain models* for automated planning. *Machine Learning* applied to *APS* has attracted a long history of research, and we point the reader to a recent survey for a full account [Jiménez *et al.*,]. This research is aimed at two distinct areas:

- learning a domain model representing the physics of the world
- learning heuristics to make the use of a planning engine more efficient.

Here we are primarily concerned with the first area, where significant progress in the automated or semi-automated acquisition of domain models has been made. Most approaches are aimed at learning representations of actions in enough detail so that artificial agents can perform deliberative reasoning with them, and in particular they can be used as inputs to an AI planning engine. Using techniques from the field of Machine Learning, researchers have experimented with processes that input training or observation inputs, and output solver-ready models in languages such as PDDL.

To be successful, these processes have to embody general properties and constraints about actions and objects, and in most cases the kind of domain in which they are learning. The key idea within these approaches is that of *inductive generalization* - using examples of behaviors of a class of objects and generalizing these examples to a theory about the whole class of objects. In the case of planning, a natural training input would be a set of plans that are observed from the domain itself. These training plans would therefore be considered part of the requirements specification. Examples of potential training plans are as follows:

- logs of commands such as operating system instructions
- logs of web service calls
- moves made in a game
- traces of workflow or business process execution

For example, GIPO III embodied an induction technique to aid the acquisition of detailed operator schemata descriptions, called *opmaker*. The tool

requires an initial structural description of the domain to work - "static" knowledge about states of objects and their relations. Given a training problem instance and a valid solution plan for that instance, *opmaker* derives a set of generalized operator descriptions resulting in a full PDDL model [McCluskey *et al.*, 2010]. This example illustrates the separation of domain model learning into three concerns:

> (i) what language is the learned domain model going to be expressed in?
> (ii) what inputs (training plans, observations, constraints, partial models etc) are there to the learning process?
> (iii) what stage is the learning taking place - initial acquisition, or incremental, online adaptation?

In the case of *opmaker*, (i) was PDDL version 2.1, (ii) was a partial model and one example, and (iii) was initial acquisition. In fact, for much of the work done up to now the answer to (i) is "some variant of PDDL forming a domain model that can be input to planning engines" and to (iii) is initial acquisition. However, adaptation can be viewed as a non-monotonic special case of initial acquisition, where input to the learning process includes the current domain model as well as training examples etc, and output is the updated model. Regarding (ii), systems that learn very expressive domain models tend to demand most detailed input, often including a partial domain model, with valid plan examples and detailed state information.

Work in learning domain models for robotic agents [Amir, 2005; Benson, 1996] assumes that a training mechanism exists with rich feedback mechanisms. Typically, much a priori knowledge is assumed, such as predicate descriptions of states, and partial or total state information before and after action execution. With such rich inputs, systems such as Amir's SLAF [Amir, 2005] can learn actions within an expressive action schema language.

In comparison, some recent work on learning domain models has concentrated on learning from example plans but with little or no input domain knowledge. The LAMP system [Zhuo *et al.*, 2010b] can form simple PDDL domain theories from example plan scripts and associated initial and goal states only. LAMP [Zhuo *et al.*, 2010a] is a successor of the ARMS algorithm, and designed to learn complex structured domain models containing quantifiers and logical implications. LAMP requires as input a set of observed plan traces, a list of actions composed of names and parameters, and a list of predicates with their corresponding parameters. It is aimed at learning in a mixed initiative fashion: the authors acknowledge that a model learned by LAMP needs to be refined by experts in order to produce a final domain model.

One problem with systems such as LAMP is in their evaluation. In LAMP's case, a very simple metric is used: the authors define the error rates of their algorithm as the number of different predicates in either the preconditions or the effects of an operator schema, as a proportion of the total number of predicates. If for example, it was required to show that with more examples, LAMP would be more accurate, then it would be necessary to show that one model was "nearer" the hand crafted version than another. Even with predefined predicates, it could be argued that the evaluation of LAMP is problematic in that is it not obvious how "near" the models it produces are using such syntactic distinctions. This measure does not seem appropriate in the situation where the system learns predicates. Measures are needed that obey certain rules - for example a measure that is monotonic in the sense that as the error rate was lower, the domain created was "nearer" the original domain.

There have been several other notable developments in learning in uncertain or partially known domains. *Reinforcement learning*, traditionally used in single goal or policy learning planners, has recently been developed for symbolic or relational learning, though its potential for learning full models of the PDDL variety is not yet proven [Jiménez *et al.*,]. A promising approach towards learning incomplete and uncertain domain models is ongoing in the *Model-lite* project [Yoon and S.Kambhampati, 2007]. Here the authors use probabilistic logic as the basis for the language of the learned domain model.

The LOCM System

We focus in this section on *LOCM*, a system which fully automatically creates a domain model from training example plans. To investigate this, we consider the following snippet of a plan training example from a "wheel change" planning function:

> open(c2); putaway_wrench(wr1,c2); close(c2); open(c1);
> fetch_jack(j1,c1); fetch_wrench(wr1,c1); close(c1); open(c2);
> fetch_wrench(wr2,c2); fetch_jack(j2,c2); close(c2);

Typically, we assume that the first symbol in each expression is the name of the action, and this stays constant through the trace, the other symbols in each element are names of "objects" affected or needed in some way by the action, and examples are presented in a consistent format, hence the same role is played by every object in the same position of the parameter sequence.

Scripts are input in the form given in the example above, where actions are referred to by action-name and assumed act on the objects referred to in

the object-list - they either change the object's state, or leave it where it is. Objects are assumed to be instances of sorts, where sorts behave the same when acted on by actions.

As with ARMS, LOCM outputs a planning domain theory in a PDDL format, but it inputs *only* plan scripts - such as the example given above - it does not require representations of initial and goal states, or any descriptions of predicates, object classes, states etc. Rather, it is assumed that objects referenced in the plan scripts are acted on by actions, in particular:

- Different instances with the same action name induce classes of objects, so for example c1 and c2 are in the same class, as they appear in the same position (1st) after the same action name (open).
- Consecutive actions acting on the same object help form behavior machines for all objects in a sort. For example, we can assume that the output state of c2 after action "open" is the same as the input state of action "putaway_wrench"
- Where consecutive actions act on the same set of (2,3 or more) objects, LOCM induces relations between object classes. For example, if in all examples of actions put_away_jack(x,y); .. ;fetch_jack(x,z); ..., where x is not referenced between the two actions, have y = z, then LOCM deduces that a relational predicate between the sort of x and the sort of y is true at the state of x after put_away_jack.
- Where subsets of a sort's objects appear in certain slots of actions, then a "static relation" is added to the state on the object.

With these assumptions, and sufficient examples, LOCM can induce a domain model in PDDL. LOCM's most exciting result is to be able to induce a complete domain theory from observing human Freecell game scripts. The domain theory was input to a general planner, which was then capable of playing Freecell.

2.3 Planning Problem Reformulation

Modeling planning tasks is not only about capturing capabilities of autonomous entities in order to provide them sequences of actions they can execute. Models should be also efficient in terms of being easily solvable by planning engines. However, there are often differences between realistic and efficient models. Therefore, there is a need for a concept which bridges a

gap between realistic and efficient models of planning tasks. This concept has been introduced in [Chrpa *et al.*, 2012a] which provides reformulation schemes consisting of planning problem reformulation function, which can be used to obtain an efficient model for the realistic one, and plan reformulation function, which can be used to obtain a solution of the realistic model from the efficient one. This basically provides a *black-box* approach which works as follows:

1. Reformulate a planning problem given at the input.
2. Solve the reformulated planning problem by a common planning engine.
3. Reformulate the solution of the reformulated problem.
4. Return the reformulated solution, which is a solution of the given planning problem.

Creating macro-operators, which encapsulate sequences of primitive planning operators, is a well known approach to reformulation which in some cases can speed up plan generation considerably [Newton *et al.*, 2007; Botea *et al.*, 2005]. Moreover [Chrpa, 2010] eliminates potentially useless primitive planning operators replaced by a generated macro-operator, however it might cause that some reformulated problems become unsolvable. Macro-operators can be understood as 'shortcuts' in the state space which might eventually result in reduction of computational complexity [Korf, 1985]. However, a disadvantage of macro-operators is in a huge number of their instances which causes an increase of the branching factor. The branching factor may be reduced by recently introduced technique [Chrpa and Barták, 2009; Chrpa and McCluskey, 2012], *entanglements*, which stand for relations between planning operators and predicates in terms of exclusive 'production' or 'requirement' of predicates.

Post-planning plans optimization can be understood as a specific form of reformulation technique. In this case plans, often produced by planners which can retrieve solution in a little time but in very low quality, can be optimized in a post-processing step. There are different strategies, for instance, exploring state space around plans in order to find shorter (and more optimal) plans [Nakhost and Müller, 2010] or determining redundant actions that can be removed from plans [Chrpa *et al.*, 2012b] or replacing sequences of actions by shorter ones [Estrem and Krebsbach, 2012; Chrpa *et al.*, 2012c].

Chapter 3

Simulation Environments

In this section we will describe some of the work carried out at AIAI in Edinburgh investigating the use of planning technology and representations to support collaboration in simulated, virtual world environments. This has been applied in the context of emergency response where procedural knowledge often corresponds to the kind of knowledge held in the representations described above.

3.1 A Virtual Collaboration Environment

Emergency situations usually call for quick and appropriate action to minimize loss of life and property. However, knowing what these actions should be is not always obvious, even if a current, post-disaster situation were known. One way to prepare for emergencies of a given type is to develop so-called Standard Operating Procedures (SOPs), manuals containing procedural knowledge describing courses of action that should be followed in a given situation. These SOP manuals represent best-practice knowledge, and are usually written by one or more experts with extensive experience in the field. Such procedural knowledge can be used to train emergency managers, for example. There is a significant amount of procedural knowledge for emergency response available today, mostly in the form of physical manuals and ranging in size from a few pages to several volumes. While these manuals are considered valuable where they exist, there are a number of problems with such documents in practice:

- Access time: While these manuals are useful for teaching the procedures they contain, they are usually not used during an actual emergency. This is simply because there is no time to search for information in large manuals. Emergency managers may have been through the SOPs,

but under stressful conditions options may be forgotten or steps may be omitted.

- Structure: The manuals are often well-structured, but there is no standard way of structuring these documents. An emergency manager who needs to be familiar with different SOPs deriving from different sources may thus find them confusing to use.

- Updating: procedural knowledge should be updated with lessons learned after every emergency in which they have been applied. This is a cumbersome task to perform with printed manuals, and even web-based documents offer limited support for this process.

The OpenVCE project [Hansberger *et al.*, 2010] aimed to develop an open virtual collaboration environment that facilitates collaborative work in a virtual space. This environment could, for example, be used to collaborate on the development of procedural knowledge, or it could be used during an actual emergency to manage information and courses of action. In fact, this environment contains a specific piece of software that supports these two functions: the <I-N-C-A> extension for MediaWiki. The OpenVCE space consists of two linked environments: a dynamic website and 3D space for meetings [Tate *et al.*, 2009] as shown in figure 3.1. This space links aspects of a web-based community portal built on widely available and established open-source software (Drupal and Mediawiki) with publicly accessible virtual world 3D spaces in Second Life.

To avoid all this technology overwhelming novice users who are attempting to collaborate in this space, the project has also developed the Virtual Collaboration Protocol, which is itself procedural knowledge that describes how this environment is meant to be used to deal with certain types of emergency. This protocol is itself supported by an extension to the website that guides users who are following the protocol.

3.2 Collaborative Development of Procedural Knowledge

The collaborative development of procedural knowledge can be supported in a number of ways using dynamic web technology. We have based our collaborative document editing facility that can be used to write SOP manuals on MediaWiki [Barrett, 2009]. The reasons for this choice are simple: MediaWiki is open-source (a project requirement), scalable (it powers Wikipedia),

Figure 3.1: OpenVCE Website and 3D Virtual Meeting Space

and there is an active community behind it. However, wiki articles are not structured to support procedural knowledge, which is why we have developed an extension that allows for the structuring of an article according to the principles underlying Hierarchical Task Network (HTN) planning, which provides a 'natural' way of decomposing tasks into sub-tasks, and as such is the structure found in many existing SOP manuals. A system that uses a similar approach, namely, representing procedural knowledge in a Wiki is CoScripter [Leshed *et al.*, 2008]. However, their representation is not based on AI planning and thus does not support the automated composition of procedures. The Incidone system [Lijnse *et al.*, 2012] uses Task-Oriented Programming to represent and use procedural knowledge in emergency response, but the representation is closer to the specific programming language used.

Bibliography

[Albore *et al.*, 2010] Alexandre Albore, Héctor Palacios, and Hector Geffner. Compiling uncertainty away in non-deterministic conformant planning. In *ECAI*, pages 465–470, 2010.

[Allen *et al.*, 1990] James Allen, James Hendler, and Austin Tate, editors. *Readings in Planning*. Morgan Kaufman, 1990.

[Amir, 2005] E. Amir. Learning partially observable deterministic action models. In *Proc. IJCAI 05*, pages 1433–1439, 2005.

[Barreiro *et al.*, 2012] Javier Barreiro, Matthew Boyce, Minh Do, Jeremy Frank, Michael Iatauro, Tatiana Kichkaylo, Paul Morrisand James Ong, Emilio Remolina, Tristan Smith, and David Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proceedings of the 4th International Competition on Knowledge Engineering for Planning and Scheduling, ICAPS 2012, Atibaia, Sao Paulo, Brazil*, 2012.

[Barrett, 2009] D. Barrett, editor. *MediaWiki*. OReilley, 2009.

[Benson, 1996] S. Benson. *Learning Action Models for Reactive Autonomous Agents*. PhD thesis, Stanford University, 1996.

[Bernard *et al.*, 2000] D. E. Bernard, E. B. Gamble, N. F. Rouquette, B. Smith, Y. W. Tung, N. Muscettola, G. A. Dorias, B. Kanefsky, J. Kurien, W. Millar, P. Nayal, K. Rajan, and W. Taylor. Remote agent experiment ds1 technology validation report. Technical report, Ames Research Center and JPL, 2000.

[Bernardini and Smith, 2007] Sara Bernardini and David E. Smith. Developing domain-independent search control for europa2, 2007.

[Botea *et al.*, 2005] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

[Bylander, 1994] T. Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69:165–204, 1994.

[Chrpa and Barták, 2009] L. Chrpa and R. Barták. Reformulating planning problems by eliminating unpromising actions. In *Proceedings of SARA 2009*, pages 50–57, 2009.

[Chrpa and McCluskey, 2012] Lukás Chrpa and Thomas Leo McCluskey. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of ECAI*, pages 240–245, 2012.

[Chrpa *et al.*, 2012a] L. Chrpa, T. L. McCluskey, and H. Osborne. Reformulating planning problems: A theoretical point of view. In *Proceedings of FLAIRS*, pages 14–19, 2012.

[Chrpa *et al.*, 2012b] Lukás Chrpa, Thomas Leo McCluskey, and Hugh Osborne. Determining redundant actions in sequential plans. In *Proceedings of ICTAI*, 2012. to appear.

[Chrpa *et al.*, 2012c] Lukás Chrpa, Thomas Leo McCluskey, and Hugh Osborne. Optimizing plans through analysis of action dependencies and independencies. In *Proceedings of ICAPS*, pages 338–342, 2012.

[Chrpa, 2010] Lukas Chrpa. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review*, 25(3):281–297, 2010.

[Coles *et al.*, 2008] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. Planning with problems requiring temporal coordination. In *AAAI*, pages 892–897, 2008.

[Currie and Tate, 1991] Ken Currie and Austin Tate. O-Plan: The open planning architeture. *Artificial Intelligence*, 52:49–86, 1991.

[Dave *et al.*, 2003] Rakesh Dave, Subhashini Ganapathy, Mary Fendley, and Sundaram Narayanan. Dynamic path planning of ground robots and uninhabited aerial vehicles in human search and rescue missions. In *Proceedings of IICAI 2003*, pages 315–322, 2003.

[David E. Wilkins, 1990] David E. Wilkins. Can AI Planners Solve Practical Problems. *Computational Intelligence Journal*, 1990.

[Dvořák and Barták, 2010] Filip Dvořák and Roman Barták. Integrating time and resources into planning. In *ICTAI (2)*, pages 71–78, 2010.

[Estrem and Krebsbach, 2012] Sam J. Estrem and Kurt D. Krebsbach. Airs: Anytime iterative refinement of a solution. In *Proceedings of FLAIRS*, pages 26–31, 2012.

[Fox and Long, 2006] Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)*, 27:235–297, 2006.

[Frank and Jónsson, 2003] Jeremy Frank and Ari K. Jónsson. Constraint-based attribute and interval planning. *Constraints*, 8(4):339–364, 2003.

[Gerevini *et al.*, 2004] A. Gerevini, A. Saetti, and I. Serina. Planning in pddl2.2 domains with lpg-td. In *Proceedings of the fourth IPC*, 2004.

[Ghallab *et al.*, 1998] M. Ghallab, C. Knoblock Isi, S. Penberthy, D. E Smith, Y. Sun, and D. Weld. Pddl - the planning domain definition language. Technical report, 1998.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated planning, theory and practice*. Morgan Kaufmann Publishers, 2004.

[Hansberger *et al.*, 2010] J. Hansberger, A. Tate, B. Moon, and R. Cross. Cognitively engineering a virtual collaboration environment for crisis response. In *Proc. ACM Conference on Computer Supported Cooperative Working*, 2010.

[Helmert, 2003] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Jiménez *et al.*,] S Jiménez, T. De la Rosa, S. Fernández, F. Fernández, and D. Borrajo. A review of machine learning for automated planning. *The Knowledge Engineering Review (in press)*.

[Kautz *et al.*, 2006] H. Kautz, B. Selman, and J. Hoffmann. Satplan: Planning as satisfiability. In *Proceedings of the fifth IPC*, 2006.

[Kissmann and Edelkamp, 2008] Peter Kissmann and Stefan Edelkamp. Gamer: Fully-observable non-deterministic planning via pddl-translation into a game. In *Proceedings of the sixth IPC*, 2008.

[Korf, 1985] R.E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[Leshed *et al.*, 2008] G. Leshed, E. Haber, T. Matthews, and T. Lau. Coscripter: Automating and sharing how-to knowledge in the enterprise. In *Proc. International Conference on Human-Computer Interaction (CHI)*, 2008.

[Lijnse *et al.*, 2012] B. Lijnse, J. Jansen, and R. Plasmeijer. Incidone: A task-oriented incident coordination tool. In *Proc. International Conference on Information Systems for Crisis Response and Management (ISCRAM)*, 2012.

[McCluskey *et al.*, 2010] T. L. McCluskey, S. N. Cresswell, N. E. Richardson, and M. M. West. Action Knowledge Acquisition with Opmaker2. In *Agents and Artificial Intelligence*, volume 67 of *Communications in Computer and Information Science*, pages 137–150. Springer Berlin Heidelberg, 2010.

[Muscettola *et al.*, 1998] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103:5–47, 1998.

[Nakhost and Müller, 2010] Hootan Nakhost and Martin Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proceedings of ICAPS*, pages 121–128, 2010.

[Nau *et al.*, 1995] Dana S. Nau, Satyandra K. Gupta, and William C. Regli. Ai planning versus manufacturing-operation planning: A case study. In *Proceedings of IJCAI 95*, pages 1670–1676, 1995.

[Nau *et al.*, 2001] D. Nau, H. Munoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 425–430. Morgan Kaufmann, 2001.

[Newton *et al.*, 2007] Muhammad Abdul Hakim Newton, John Levine, Maria Fox, and Derek Long. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS 2007*, pages 256–263, 2007.

[Pease and Carrico, 1996] R. Adam Pease and Todd M. Carrico. Object model working group core plan representation. Technical Report AL/HR-TP-1996-0031, United States Air Force Armstrong Laboratory, Wright-Patterson AFB, OH, 1996.

[Richter and Westphal, 2010] S. Richter and M. Westphal. The lama planner: guiding cost-based anytime planning with landmarks. *Journal Artificial Intelligence Research (JAIR)*, 39:127–177, 2010.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974. Reprinted in [Allen *et al.*, 1990, pages 98–108].

[Simpson *et al.*, 2001] R. M. Simpson, T. L. McCluskey, W. Zhao, R. S. Aylett, and C. Doniat. GIPO: An Integrated Graphical Tool to support Knowledge Engineering in AI Planning. In *Proceedings of the 6th European Conference on Planning*, 2001.

[Simpson *et al.*, 2007] R. M. Simpson, D. E. Kitchin, and T. L. McCluskey. Planning Domain Definition Using GIPO. *The Knowledge Engineering Review*, 22(1), 2007.

[Simpson, 2007] R. M. Simpson. Structural Domain Definition using GIPO IV. In *Proceedings of the Second International Competition on Knowledge Engineering. Providence, Rhode Island, USA.*, 2007.

[Tate *et al.*, 1994] A. Tate, B. Drabble, and R. Kirby. O-Plan2: an Open Architecture for Command, Planning and Control. In M. Fox and M. Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[Tate *et al.*, 2009] A. Tate, S. Potter, and J. Dalton. I-room: a virtual space for emergency response for the multinational planning augmentation team. In J. Lawton, J. Patel, and A. Tate, editors, *Proc. 5th International Conference on Knowledge Systems for Coalition Operations (KSCO)*, 2009.

[Tate, 1977] Austin Tate. Generating project networks. In *Proc. 5th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 888–893. Morgan Kaufmann, 1977. Reprinted in [Allen *et al.*, 1990, pages 291–296].

[Tate, 1998] Austin Tate. Roots of SPAR—shared planning and activity representation. *The Knowledge Engineering Review*, 13, 1998.

[Teichteil-Königsbuch and Fabiani, 2006] Florent Teichteil-Königsbuch and Patrick Fabiani. Autonomous search and rescue rotorcraft mission stochastic planning with generic dbns. In *Proceedings of IFIP AI 2006*, pages 483–492, 2006.

[Tran *et al.*, 2008] SDang-Vien Tran, Hoang-Khoi Nguyen, Enrico Pontelli, and Tran Cao Son. Cpa(c)/(h): Two approximation-based conformant planners. In *Proceedings of the sixth IPC*, 2008.

[Vaquero *et al.*, 2007] Tiago Stegun Vaquero, V. Romero, F. Tonidandel, and Jose Reinaldo Silva. itSIMPLE2.0: An integrated Tool for Designing Planning Environments. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). Providence, Rhode Island, USA.*, 2007.

[Wilkins, 1988] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufman, 1988.

[Yoon and S.Kambhampati, 2007] S. Yoon and S.Kambhampati. Towards model-lite planning: A proposal for learning & planning with incomplete domain models. In *Proc. Workshop on AI Planning and Learning, ICAPS*, 2007.

[Younes and Littman, 2004] Hakan L. S. Younes and Michael L. Littman. Ppddl1.0: An extension to pddl for expressing planning domains with probabilistic effects. Technical report, Carnegie Mellon University, 2004.

[Zhuo *et al.*, 2010a] Hankz H. Zhuo, Qiang Yang, Derek H. Hu, and Lei Li. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18):1540–1569, December 2010.

[Zhuo *et al.*, 2010b] H.H. Zhuo, Q. Yang, D. H. Hu, and L. Li. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18):1540 – 1569, 2010.