Contents

Knowledge Engineering for Planning ROADMAP

Editor: T. L. McCluskey
Contributors: R. Aler & D. Borrajo& P. Haslum
& M. Garagnani & P. Jarvis & T. L. McCluskey
& I. Refanidis & U. Scholz

22nd August 2003

1 Introduction

1.1 Outline Definition

We refer to *AI (Artificial Intelligence) Planning* as the science and technology of automated intelligent reasoning that involves actions, events, goals, activities, time and resources. By its very nature AI Planning is knowledge-based: planning involves the manipulation of knowledge of complex phenomena, such as actions themselves. The scope of knowledge associated with a planning application is called the *domain*, while the symbolic representation of this is called the *domain model*. Planning programs input goals or directives, and, using a domain model, find some ordering of actions to achieve them. Knowledge Engineering (KE) in AI Planning is the process that deals with the acquisition, validation and maintenance of planning domain models, and the selection and optimization of appropriate planning machinery to work on them. Hence, knowledge engineering processes support the planning process: they comprise all of the off-line, knowledge-based aspects of planning that are to do with the application being built, and any on-line processes that cause changes in the planner's domain model.

1.2 Domain Modelling

The main characteristic of a domain model is that it is possible for an agent to use one to make rational deductions about the domain it represents. In particular to planning, it is commonly assumed that a model contains a declarative description of domain dynamics and that the model's most important component is a set of action descriptions. The conventional wisdom is that a declarative model should be developed to a certain extent independently of the planning engine and any other software that will form the rest of the application. This tends to ease the process of validating the domain model. It also gives the stakeholders and developers more flexibility in the use of their product and lessens the investment risk; an independent domain model may be used with a range of general planning engines, and may be used in many other ways not necessarily connected to planning.

Acquisition of a domain model may involve a prolonged analysis of the application domain by knowledge engineers, using structured interviews with stakeholders, system manuals, existing models, software documentation etc. Before acquisition can take place, however, the engineer must find out what kind of planning problem is to be solved by the system. This entails asking questions such as: Is the problem one of scheduling or planning, or a combination of the two? Can actions be modeled deterministically? Do they have to be modeled with duration? Do we have to model the resource they consume? Can we assume the planner has complete knowledge of the world? What are the kind of plans required? What is the relation between planning and execution? What are the the optimization criteria for plans? Hertzberg in section 3 of reference [39] declares a similar list of questions in his discussion of the characteristics of application domains.

The results of this early analysis is fundamental to and implicit in the kind of domain modelling language that will be used to capture the domain model, as well as critical in the choice of planning algorithm. Once the type of planning application has been established, and environmental assumptions are known, the process of domain modeling itself can begin. The knowledge acquired tends to fall into two categories:

(1) Domain structure and dynamics: The identification of the relevant objects and object classes in the domain, their properties and relations (predicates), and constraints among these predicates. Dynamic knowledge is dependent on (1), but typically involves specifying how these predicates change truth value as actions are executed.

(2) Domain heuristics: General approximate rules and the general applicability of techniques that have been found to help in plan generation, where their use leads to a speed-up in planning and/or an increase in the quality of the output plans.

From an engineering point of view, it is essential to be able to **validate** the results of domain acquisition. Validation of a domain model is the process that promotes its quality in terms of internal and external criteria by the identification and removal of errors in the model. Internal criteria include properties such as syntactic correctness and logical consistency; in general these properties can be proved formally and are not problematic. External criteria include properties such as accuracy, correctness and completeness. Given that the sources of the model will not often be a mathematical object, these properties can never be proved correct (in the same sense that a requirements specification can never be proved correct). Note the distinction between validation of a domain model and validation of a planning system. The former supports the latter, and occurs at a much earlier stage in system development.

1.3 The Agents Involved in the Knowledge Engineering Process

There are several kinds of roles that have to be filled in the technical side of the knowledge engineering process: planning experts, domain engineers, domain experts, software and HCI experts, and end users. Often a person may have more than one role; a researcher in academia may have to fill every role. Assuming there is a distinguished formal system within which the domain model is being encoded, it seems necessary to have several kinds of "interface languages" to this formalism to suit participants in different roles. You would not want, for example, a user to help in the static validation of a domain model by reading through a complex logic formalism.

1.4 The Bigger Picture

Knowledge acquisition for knowledge-based systems and requirements engineering in software engineering are very active areas in Computer Science, and are associated with a growing body of literature, methods and techniques. They are related areas in a number of respects. In particular, one can consider that the results of knowledge acquisition and requirements engineering both involve some kind of domain model [77]. Using our general definition above, KE for planning may be seen as a special case of these general areas. Hence it may prove useful to derive methods and adapt tools from these areas, although there are peculiarities of planning that clearly distinguish engineering planning knowledge from them:

- The knowledge elicited in planning is largely knowledge about actions and how objects are effected by actions. This knowledge has to be adequate in content (and ultimately in form) to allow efficient automated reasoning and plan construction. In contrast, knowledge elicited about processes/actions in traditional software engineering tends to be done with the purpose of helping in the analysis and understanding of a system, and to be used in the forming of a specification of a new system.
- The ultimate use of the planning domain model is to be part of a system involved in the "synthetic" task of plan construction. This makes it very specific in the world of Knowledge-based Systems

(KBS), where many successful systems are, in contrast, aimed at solving diagnostic or classification problems.

Despite the difference of purpose, adopting tools and techniques from these general areas seems likely to be a good strategy. For example, the insights gained from the use and development of methods such as KADS [2], and the use of requirements modeling languages and methods in software engineering (e. g. [33]) need to be used when developing KE methods for planning applications.

The number of planning projects which have exploited KE tools and techniques is growing rapidly. Some examples of planning projects that have addressed or are addressing the issues of KE and/or knowledge representation are as follows:

- O-Plan, SPAR and <I-N-OVA>: Edinburgh/ARPI projects that tackle plan representation and KB planning (e.g. see [80, 81])
- The Planform project, aimed to create a simple knowledge engineering tools environment (e.g. see [1])
- The SIPE-2 system, featuring advanced HCI tools (e.g. see [66])
- The multimission Vicar Planner applied to automated image processing
- The Remote Agent Experiment, the control of NASA's Deep Space 1 by an AI planner.

Experience in applied work in planning suggests that the outputs of the KE process, such as "activity descriptions", are a vital core concept for many types of reasoning beyond planning. KE tools, methods and languages need to be suited to the much wider role of capturing knowledge of activity whether or not planning is involved in guiding and directing those activities towards purposeful outcomes.

2 Knowledge Representation

2.1 Introduction

The kinds of knowledge that need to be captured in order to build a planning application include knowledge about actions, goals, activities, time and resources. So called 'classical' planning has adopted a propositional state-based approach to knowledge representation, where states of the world are modelled by statements in logic, and actions are modelled by transformations between these states. Currently the research community's standard for communicating domain models is PDDL [30, 20]. This was not, however, designed from a knowledge engineering perspective [59]. In the sections below we discuss what kind of requirements exist for practical representation languages, and introduce alternatives to traditional propositional encodings.

2.2 Requirements for a Representation Language

How can we provide better formalisms for capturing and expressing the required knowledge? Evidence from the KBS and Requirements Engineering Communtity suggests that a domain modeling language should:

- Be structured. It should provide mechanisms that allow complex actions, complex states and complex objects to be broken down into manageable and maintainable units. For example, the dynamic state of a planning application could be broken down into the dynamic state associated with each object. On this structure can then be hung ways of checking the model for internal consistency and completeness.

- Be associated with a workflow model, or method. This will give a set of ordered steps to be carried out in order to capture the domain model, thus guiding the knowledge engineer throughout the process. It could contain activities such as modeling of state changes and the discharging of proof obligations.
- Support the operational aspects of the model. The language's framework should include a set of properties and metrics which can be evaluated to assess a model's operationality and likely efficiency. It should be possible to predict whether the model can be translated to an efficient application.
- Be tool supported. These tools will support the steps in the method and, using the structure of the model language, be able to provide powerful support for statically validating, analyzing and operationalizing the model.
- Be expressive and customizable. The language needs to be generally applicable, yet customizable in some sense so that it "fits" well with applications. Since there is a whole range of assumptions involved in planning which may or may not hold in an application (e.g. to do with uncertainty, resources, closed world) it may be that the modeling language will have "variants" to deal with different assumptions.
- Have a clear syntax and semantics. As a basis for the other aspects above, and for the analysis of models encoded in the language, it should be possible to map models to a "meaning" within some well-known formal system such as a modal logic.

It seems that no modelling language fulfills all these criteria. Languages that have been developed from the point of view of knowledge acquisition include DDL.1 [15], TF [81] and OCL [1]. The most commonly used domain description language, PDDL [30], appears to fare badly according to these criteria. PDDL, however, has the potential to be easily extensible and at the same time widely accepted and used within the planning community.

2.3 The domain representation: Propositional vs. Analogical

In addressing the question of how to provide better formalisms for capturing and expressing the required knowledge, the previous section proposed a list of generic requirements for a representation language. According to such requirements, an ideal domain-modelling language should be expressive and customizable, yet be able to produce an *efficient* encoding of the problem domain. This section takes a more critical and constructive approach, by (1) arguing that most of the current planning modelling formalisms produce rather inefficient encodings of realistically complex domains, and by (2) actually showing *how* more efficient formalisms can be developed.

In particular, in the field of knowledge representation two main types of modelling paradigms have been identified: the widely used *propositional* (or 'sentential' [49]) representation, and the so-called *analogical* (or model-based) representation [8]. This section illustrates how the main inefficiencies of current planning representations (namely, the frame and ramification problems) arise from the use of purely propositional representations, and claims that such problems can be overcome through the adoption of analogical planning domain descriptions.

2.3.1 Problems associated with propositional languages

For many years, researchers in AI have regarded the frame problem [58] as presenting a major difficulty for reasoning about action [27][76]. The frame problem (in essence, having to specify, for each action, which properties of the world remain *unaffected* by its execution) can be seen as the result of adopting a specific knowledge representation paradigm, and was addressed in STRIPS by requiring that the representation of an action (operator) explicitly listed only those propositions that *are* affected by the actual execution of the represented action, while those that are unaffected are simply assumed to persist [51]. This assumption, still lying at the basis of modern planning domain description languages [20], allows the

frame problem to be avoided, but does not address a second, equally serious representational issue: the so-called *ramification* problem [27]. John Pollock [70] accurately describes this problem as one that

> *"arises from the observation that in realistically complex environments, we cannot formulate* axioms *that* completely specify *the effects of actions or events.* [...] [I]*n the real world, all actions have infinitely many ramifications stretching into the indefinite future. This is a problem for reasoning about change deductively* [...]"[70, p.536]

To illustrate the significance and impact of this problem in planning, let us consider a variation of the familiar Blocks-World (BW) domain, in which the robot arm can lift towers of m blocks, with $m \leq h \in N$. In order to decide which blocks may be lifted in a given state, it should be possible to augment the problem description with the two following domain axioms:

$$\forall x, y : \texttt{on}(x, y) \rightarrow \texttt{above}(x, y, 1) \tag{1}$$

$$\forall x, z \ (\exists y, n : \texttt{on}(x, y) \wedge \texttt{above}(y, z, n) \rightarrow \texttt{above}(x, z, n + 1)) \tag{2}$$

The condition "$\texttt{clear}(x) \wedge \texttt{above}(x, y, n) \wedge (n \leq h)$" would then be sufficient to guarantee that 'y' is the lowest block of a 'liftable' tower. Unfortunately, no one has yet provided a semantics for domain axioms containing numeric fluents, such as those shown above. Most importantly, given a certain world state (described using exclusively '`on( )`' expressions), deducing the '$\texttt{above}(x, y, n)$' relations for all the possible blocks requires a number of axiom applications that grows exponentially in the size (arity) of the axioms. More precisely, if 'o' is the number of objects (constant symbols) of the domain and 'r' is the arity of the axioms, the number of deductions required is in the order of $k \cdot o^r$, for a given constant k.

Because of this, extending planners to deal with domain axioms seems likely to involve significant additional computational costs. Consider, for instance, how a forward state-space planner could solve a problem in the above BW domain with axioms (1) and (2): whenever the truth value of any of the '`on( )`' propositions changes, it will be necessary for the planner to re-calculate all of the '`above( )`' relations, as the truth of some of them will have been affected by the change. A backward-search planner would incur in similar problems. Consider, for example, the process of determining how to achieve the goal (or precondition) '$\texttt{above}(x, y, j)$', in which the variables x and y are still unbound (i.e., how to build a tower of $j + 1$ blocks, for a given j, using any set of blocks). Transforming this expression into the equivalent formula "$\texttt{on}(x, z_1) \wedge \texttt{on}(z_1, z_2) \wedge \ldots \wedge \texttt{on}(z_{j-1}, y)$" (which can then be achieved using the standard BW operators) requires a rather complex process of reasoning, analogous to that used in the resolution procedure of a Prolog interpreter. In addition, in a backward-chaining search, the presence of axioms would significantly increase the branching factor.

In view of these difficulties, several researchers (e.g., [23][26][17][82]) have been exploring the possibility of *preprocessing* planning problems and compiling axioms away in the domain description. However, recent theoretical results [82] show that this approach leads (in general) to exponentially (or polynomially, if the arity of the axioms is kept constant) longer domain descriptions and *plans*. Experimental evidence suggests that the resulting performances can be even worse than those obtained with planners that are able to deal with domain axioms internally [82].

To summarize, the ramification problem and the presence of domain axioms add a significant computational overhead to the planning process. This may become severe or even unacceptable if planning is to be carried out in realistically-complex scenarios, rather than in simple toy domains. In fact, when reasoning about (i.e., simulating) action in real-world domains, innumerable physical properties of the world should be taken into account, such as sound and light propagation, temperature, gravity, fluid and gas dynamics. Using Pollock's original example, among the effects of striking a match we must include such things as "displacing air around the match, marginally depleting the ozone layer, raising the temperature of the earth's atmosphere, marginally illuminating Alpha Centauri, making that match unavailable for future use, etc." [70, p.537]. In realistic domains, the number and complexity of the axioms quickly become overwhelming, making planning very difficult a task.

It is contended here that the problem of domain axioms and that of frame axioms (requiring, respectively, the explicit specification of all properties that are - and that are not - affected by action) are,

ultimately, two facets of the same *representational* problem, which lies in the use of purely "propositional" (or 'sentential', or 'Fregean' [48]) domain description languages.

The term 'propositional' is used here as opposed to 'model-based' [34], or 'analogical' [79][18], or 'direct' [8][50], and indicates a representation in which the current world state is described by a set of Well-Formed Formulæ (sentences) of a formal language (e.g., predicate logic). While a propositional representation can be said to *describe* the domain represented, an analogical representation rather *models* it [48]. In particular, in model-based (or analogical) representations the world is modelled by data structures (and relations on them) which mirror the properties of and relations between the objects of the represented domain (in the next section a more detailed definition is provided).

Being purely descriptive, propositional planning languages (such as STRIPS, ADL and PDDL) are quite flexible and expressive (e.g., see [20]). Yet, because of their descriptive nature, they require all (physical) properties and constraints of the world (even the most trivial, such as the fact that an object cannot be moved on top of itself) to be represented 'extrinsecally' [69], i.e., to be explicitly imposed on the model via the addition of supplementary elements (in the form of formulæ and axioms). As seen earlier, this can quickly lead to a combinatorial explosion in the number of (trivial) inferences that must be explicitly represented and carried out – namely, to the frame and ramification problems. Hence, although propositional languages are widely applicable and may be used to model most aspects of the real world, they tend to produce representations that are *inefficient* for domains that involve a large number of distinct entities subject to (even simple) physical constraints.

In order to overcome these problems, more adequate planning domain description languages are needed. The author of this section (Max Garagnani) maintains that one of the ways in which new, more efficient (yet expressive) representations can be developed lies in the introduction of analogical formalisms, and in their use in conjuction with propositional descriptions.

2.3.2 Planning with analogical representations

Planning in realistic domains is closely related to the problem of common-sense reasoning [57]. In this context, several researchers (e.g., [48][50][49][45]) have argued for the need of formalisms that allow a more direct (or 'vivid') representation than current sentential descriptions. In particular, analogical and diagrammatic representations (briefly introduced earlier) have long been of interest to the knowledge representation community (e.g., [79][38][65][52] – see [48] for a useful account). In order to better characterize such representations, let us introduce a simple example of analogical representation, which will be used throughout the rest of this section.

Consider a representation of the Blocks World domain in which the state is modelled as a set of lists of characters. Each list denotes a (possibly empty) stack of blocks, and each character represents a block. For instance, in a four-block problem, the state S = {[A,B,C],[D],[ ],[ ]} would correspond to the propositional description {`on`(C,B), `on`(B,A), `clear`(C), `clear`(D)}, in which predicates have their standard meaning (empty lists denote empty spaces on the table). The specification (using an appropriate syntax) of a generic operation for transforming legal states into legal states (consisting of removing the last character of a non-empty list and appending it to another - possibly empty - list) completes the description of the domain model.

Using this simple example of analogical representation for the (one-operator) BW domain it is already possible to identify the main features that differentiate such representations from propositional ones.

In a propositional representation the objects of the domain are represented as constant symbols (e.g., A, B, C,... etc.). The relevant relationships (like 'on' and 'above') between objects are described as sets of associations (relational instances) between such symbols, and are identified using other symbols (predicates) of the language (e.g., `on`(B,A), `above`(C,A)). The specific syntax chosen to build such formulæ, however, has no bearing to the properties of the represented world. In particular, the properties of the relations that exist between the objects of the domain and their interactions are *imposed* on the formal model through the specification of axioms and logical rules (e.g., axioms (1) and (2)).

In contrast, in an analogical representation the objects of the domain and the relationships that exist

between them are not described by sets of relational instances, but modelled using appropriate data structures. The *syntax* of such data structures mirrors, for the relevant aspects, the semantics (relations and properties) of the problem domain [8]. In particular, the relations between elements of the representing structures of the model and the corresponding represented relations of the domain have the same algebraic structure (using Palmer's term, they are "naturally isomorphic" [69]).

Consider the BW domain example. The analogical, list-based representation consists of a set of formulæ having the following syntax:

$$[arg_1, arg_2, \ldots, arg_n]$$

This syntax clearly reflects the semantics of BW: the first character of the list (arg_1) always represents the lowest block of a stack, while two consecutive characters arg_k, arg_{k+1} indicate that block arg_{k+1} is on block arg_k. The rightmost character of a list denotes a 'clear' block. In contrast, the formulæ used in the propositional representation adopt the following generic syntax:

$$predicate(arg_1, arg_2, \ldots, arg_n)$$

This syntax has no direct relation with the structure and properties of the BW domain.

In addition, consider the spatial relation "above", represented in the model by the relation "to the right of", defined on the (linearly ordered) characters of a list[1]. The *transitive* property of the relationship "above" (originally imposed on the model through the addition of axioms (1) and (2)) is an implicit property of the relation "to the right of", and does not need to be explicitly imposed or accounted for during the reasoning process. In other words, the latter relation and the represented spatial relation "above" are naturally isomorphic [69].

Thanks to the above features, analogical descriptions can implicitly embody *constraints* that sentential representations would normally have to make explicit: the relevant properties of the relations existing between the objects of the domain are *inherent* to the structure of the representing relations, and do not need to be explicitly imposed on the model or included in the description.

Because of the implicit, unalterable structure of the relations which they employ, however, analogical representations tend to be less flexible and general than propositional ones. Nevertheless, the presence of such "inherent constraints" [69] reduces the computational complexity of inference, avoiding the combinatorial explosion of trivial deductions that would have to be explicitly represented in sentential reasoning systems, and easing the frame and ramification problems (cf. [52]). In the BW example, given a certain state description, the problem of deducing – using axioms (1) and (2) – whether a tower is 'liftable' becomes one of simply checking the current model, by counting the number of characters that follow a given one in a list. This check can be carried out in time *linear* in the number of blocks. In other words, thanks to the model-based nature of the representation, the computationally expensive theorem-proving aspects of the planning process can be replaced by simple model-checking (see also [34][45]).

By allowing simpler and more efficient methods of common-sense reasoning, analogical representations also make possible new, easier heuristic extraction, abstraction, and learning techniques. The data structures that form the model of the domain can be quickly and effectively checked for the presence of specific (syntactical) structures (or 'patterns') of objects; the recurrence of such patterns can then be used as a basis for macro-operator learning, abstraction, or evaluation metrics. For example, consider the difficulty of determining the existence of identical stacks of blocks in two different BW states: in a list-based representation, simple pattern-matching will quickly lead an answer, while a propositional encoding would involve a significantly more complex procedure.

Naturally, the simple list-based representation used for the BW domain example is not very expressive, and much more general representations are needed for modelling realistic problems. One way to obtain more expressive languages is to develop more 'abstract' and general data structures and formalisms for analogical representations, which can be used to model a wider variety of domains. This is the approach adopted by Garagnani in [24], where a more expressive representation for physical domains is

[1] More precisely, the block denoted by character x is *above* the one denoted by y iff character x appears to the right of y (in the same list).

presented, based on the abstract structure of 'SetGraph'. This formalism can adequately and efficiently represent domains that involve object manipulations, movements and changes of state, and is at least as expressive as the classical STRIPS language. However, like STRIPS, this formalism would need to be further extended in order to be able to represent realistically complex domains, allowing such features as conditional effects, durative actions, numerical quantities and functions.

A second promising direction for developing new formalisms consists of integrating analogical and propositional representations within a single 'hybrid' paradigm. Early work on the integration of model-based and sentential representations can be found in [65][35]. For example, Myers and Konolige [65] proposed a formal framework for combining analogical and deductive reasoning. In their system, the inference rules of a sentential module (based on a logical first-order predicate language) were used to reason about and 'complement' the information contained in analogical (diagrammatic) structures, which represented the state of the world. However, the system did not allow the representation of actions for transforming the current world state.

An object-centred domain-description language (OCL) has been recently proposed by Liu and McCluskey in [53] (see also [1]), based on the idea of modelling a domain as a set of objects subject to various constraints. The idea of an object-based representation is clearly in line with a model-based approach. However, OCL is fully sentential; consequently, some of the constraints which would be implicit in an analogical representation still need to be explicitly declared (e.g., the fact that if an agent holds an object of a certain type, the location of the object must coincide with that of the agent). Nevertheless, OCL seems likely to represent a good candidate language to support the use of analogical representations in conjunction with propositional descriptions.

2.3.3 Actions for the Planning Community

The choice of which representation formalism to adopt in order to capture and express the required knowledge is clearly of vital importance for KE in planning. Research in AI has since long demonstrated that the type of representation adopted plays a fundamental role in determining the difficulty of problem solving (e.g., see [6][32][78]).

The introduction of model-based and analogical representations appears to be a promising approach for addressing the ramification problem, which arises when modelling realistically-complex domains. Hence, the recommended tasks for the research community are:

- investigate analogical, model-based and diagrammatic representation tools, techniques and methods that may be relevant to planning;
- attempt to implement more expressive and universally applicable analogical representations, providing a clear syntax and semantics of the language;
- build on existing work on the integration of analogical and propositional formalisms to realise *hybrid* planning-domain description languages (this may also be achieved by extending current propositional representations with analogical features, and vice versa).

3 Knowledge Engineering Support Tools and Environments

3.1 Tool Support

To create a non-trivial domain model a team needs tool support. Naturally, the amount and coverage of tool support required depends on the size and nature of the application. Because AI Planning has been largely in the realm of research, most existing domain models have been built using nothing more than basic syntax checkers. Researchers have tended to "debug" their domain model through dynamic testing, using the failure of plan gneration or plan execution to show the presence of, and identify, domain errors. For an application requiring a larger domain model, this approach appears at best inefficient.

For knowledge acquisition, tools are needed to support interviews with domain experts and encoding of their knowledge. Machine Learning tools that for example induce operator descriptions from traces of actions may also be used. A further possibility is to provide an interface to the model's formal language that allows a domain expert to directly encode planning knowledge. This could well be more efficient than having to employ a planning expert and knowledge engineer, and may preserve in the user the feeling of control and ownership of the problem.

Developing domain models in isolation from a planning engine means that we can split the process of knowledge acquisition into user inspection, static validation, and dynamic validation (animation). In user inspection, an appropriate tool would be one that maps back and forth between a user-friendly, diagrammatic language and the domain model language itself. This kind of tool would be similar to a graphical front-end to a formal specification language.

Tools for static validation essentially perform domain analysis: They reason with the model to check that it is self-consistent, to check that it is complete (in a restricted sense) and to output consequences of the model that might be useful for user inspection. For example, tools may check that an operator is consistent (it never inputs a valid state and outputs an invalid state); or reason with operators and output state invariants to be visually checked by a user; or output necessary goal orderings, to check for impossible goal combinations and help in dynamic testing.

Dynamic validation entails acquiring a set of test cases and associated (optimal) plans and using these to test the functioning of the planner as well as the validity of the model. Generated plans can be compared against expected plans in an attempt to identify errors in the domain model.

Both the O-Plan and SIPE projects have developed tools that help in the knowledge engineering process. With O-Plan we have the "Common Process Method" [71, 81]. With SIPE we have the Act Editor [66]. Both are essentially sophisticated directed graph presentation tools, i. e. they let the developer enter domain knowledge, and examine the nodes in a HTN operator and the temporal constraints between them. Deficiencies of the O-Plan/SIPE visual tools is that (1) they provide no visualization or checking for the state based model that the user builds around the operators. There is no definition of what properties the operators available for refining a task must hold nor the transitions that a domain object can pass through or the states that it can exist in. (2) There has been no evaluation of these tools to determine if they help domain model development and to what extent. (3) None of the tools are appropriate for use by domain experts without a computer background.

Given the importance of engineering methods in other disciplines, one can argue that the development of *knowledge engineering methods* for AI Planning, based around a set of tool support, is desirable. However, it is difficult to to evaluate and benchmark knowledge engineering methods. It appears very expensive in terms of time and effort to carry out case studies, and even then we only have the results of the case. It appears that the only other way is to compare a method to existing methods in similar domains, or base a KE for planning method on an existing one e. g. a planning-oriented form of KADS. It could be argued that the general problem of method evaluation tends to make researchers avoid looking into KE as it is harder to publish (given the problems of evaluation of the research).

3.1.1 Supporting Large Scale Planning Knowledge Bases

Effective tools for supporting knowledge engineers in constructing large-scale planning knowledge bases are at last emerging from the research community. The goal of this section is to draw on the experience of an experienced knowledge engineer (Peter Jarvis of SRI) to encourage promising directions, dissuade a couple of approaches, and add an important class of knowledge engineering tasks to the research agenda.

Keep Providing Structure

"The knowledge bases that I work with contain of the order of 100 - 200 action schemata ranging over 5 abstraction levels and are described using a conventional hierarchical task network formalism. Using action schemata to structure a domain model results in an opaque description where domain properties are scattered, duplicated, and frequently implicit.

Emerging modeling approaches that encourage a central description of each object in a domain together with the state collections that each can occupy should be encouraged. It is encouraging to see new structural notions emerging that allow properties of action abstraction levels to be specified.

To exploit these developments in real domains I need them to work with existing applied planning engines. I currently have to balance the benefits of an improved modeling framework against the cost of loosing the broad inference range and efficiency of the engines I work with."

Please, No More Cosmetic Structured Editors

"Simple structured editors have often been produced to answer client knowledge engineering concerns. They typically provide attractive interfaces that graph the components of action schemata to show their ordering constraints and provide tree views for navigating through action abstraction layers. While these tools provide useful viewing aids, they still force domain descriptions to be organized around action schemata. The next time you see one of these tools put forward as the solution to the planning knowledge engineering problem, I encourage you to ask about the automatic consistency checks that the tool provides and the classes of error that they uncover."

Planning Knowledge Bases Need Support over a Long Lifespan

"The succinct representation of domain properties and design decisions together with automated consistency checking tools become even more important as a knowledge base ages. I give three examples of the classes of knowledge base change that I frequently make to encourage the inclusion of model evolution on the research agenda.

The simplest class of change is the addition of a new method for refining a task (we have drive and fly, add take-train). Most of the errors I introduce when making changes of this class center on me neglecting to model for the new method some transformation obligation imposed by its parent.

A more complex (but almost as frequent) class is significant structural changes to a knowledge base. For example, a stand-alone knowledge base that covers military air operations might need to be moved to become a component of a larger knowledge bases that covers air and ground operations. These changes often demand considerable changes in the underlying conceptualization of the domain that ripples through all the action schemata in a knowledge base.

Finally, the representational devices used to describe a domain will evolve as the underlying planning engine is refined and enhanced. The domain descriptions that I support frequently have had to undergo significant structure changes and continue to run on several versions of a planning engine."

3.1.2 Actions for the Planning Community

- **Support for Maintenance:** Carry out research into the kinds of tools that would be useful in maintaining large Planning knowledge bases.
- **Structured Representation Languages:** Promote the inclusion of *structural* features of planning description languages as a necessary precondition for their use as domain modelling languages.
- **Visualisation Tools:** Carry out research studies to evaluate the effectiveness of currently available visualization tools, and derive lessons that can be used in the design of future tools. Aim to design visual tools that draw on the whole of the domain model (objects, object hierarchies, invariants, states, as well as actions) and help in checking self consistency, accuracy and error removal.
- **Evaluation of knowledge engineering methods:** Develop models with which to to evaluate and benchmark knowledge engineering methods.

3.2 Domain Analysis

With present planning technology, finding heuristics and domain control knowledge to improve planning efficiency and plan quality is an important aspect of the knowledge acquisition process. Planning sys-

tems receive such knowledge in two different ways: Either it is given as input along with the problem specification, in which case it is often called advice, or a planning system employs *domain analysis*, *ie.* automatic knowledge discovery tools.

The knowledge found by domain analysis, commonly called *domain knowledge*, lies in between specification and advice. Briefly, it consists of statements about a planning problem that are logically implied by the problem specification, but that are not part of the specification. Furthermore, domain knowledge should be "planner independent", *ie.* not closely tied to the internal workings of any particular planning system, but such a requirement is difficult to formulate precisely.

The importance of domain analysis results from a rich structure commonly "hidden" in the domain description. Instead of letting a domain expert analyze this structure by hand and formulate this knowledge (or let a machine learning system discover it), it is often possible to find it automatically. In short, "if a machine can do something for you, don't do it yourself!".

DA techniques can aid planning and knowledge engineering for planning in several ways:

Planning speed-up and plan quality improvement: This has been the focus of research in domain analysis so far. It can be done either "off-line", i. e. only once for each domain, or "on-line", i. e. for every problem instance, depending on the DA technique.

Model validation: Static analysis can aid in the internal validation of a domain description, e. g. to find state invariants for user inspection, and to analyze the effects of applying hand-coded control knowledge.

Matching planning technology with domains: Domain analysis deals with structural features of planning domains and problems, and thus it can help in choosing the right planner and/or the right control knowledge for this planner in an automatic way.

There is a large variety of domain analysis techniques described in the literature, but most of them are integrated with a specific planning system and are not available as separate modules. A reason for this is that domain analysis does not directly produce control knowledge: It is only in combination with knowledge of the planning algorithm that domain knowledge can be effectively used for control. Nevertheless, most DA techniques are useful for more than one planner and hence should be recognized a separate modules.

TIM [21, 22] finds types and state invariants, as do DISCOPLAN [28] and others [74]. RIFO [67] removes irrelevant facts and operator instantiations, while RSA [75] and RedOp [36] find different types of constraints on what action sequences are necessary or relevant for solving a given problem. Detection of symmetry [22, 16] and goal ordering [47] can also speed up planning.

Some DA techniques find knowledge which is useful in combination with particular planning algorithms, e. g. STATIC [19] or Alpine [46], others are helpful for a class of them, e. g. TOP [75] for total-order planners. Several planners have some analysis preprocessing step, e. g. the graph construction with mutexes in Graphplan [12] or precomputation of heuristics [73, 14].

The use of domain analysis tools as separate modules requires a language to state the resulting knowledge. One attempt in this direction is the domain knowledge expression language DKEL [37]. The use of DKEL is demonstrated by the testbed for planning systems [86], which supports the quick construction and evaluation of planning systems from modules.

The extraction of properties such as types and invariants can be a way to automatically characterize planning domains and problems [54]. It can also be used to detect subproblems, e. g. a TSP or shortest-path problem, embedded in a planning problem [55]. Another use of domain analysis is reasoning about the planning process. An example of this is the planner HAP [87], whose planning strategy is adjusted according to the existence and characteristic of domain properties, which are in part found by domain analysis.

3.2.1 Action for the Planning Community

According to the points above, the planning community should

- Decouple domain analysis techniques from planning engines, so as to enable arbitrarily combining different DA techniques and planning engines. For this to be possible, the domain description formalism that is input to the planning engine must support expression of domain knowledge.
- Analyze hand-coded control knowledge in use and design automatic tools to find as much of that knowledge as possible.

3.3 Planning Ontologies

There are several attempts to create planning ontologies traditional applications. One of them is the Shared Planning and Activity Representation (SPAR) project (http://www.aiai.ed.ac.uk/ arpi/spar/), which is part of the DARPA/Air Force Research Laboratory (Rome) Planning Initiative (ARPI) (http://arpi.isx.com/). SPAR [80] is based on the assumption that it is important that information about processes, plans and activities is able to be shared within and across organisations. Cooperation and coordination of the planning, monitoring and workflows of the organisations can be assisted by having a clear shared model of what comprises plans, processes and activities. SPAR is intended to contribute to a range of purposes including domain modelling, plan generation, plan analysis, plan case capture, plan communication and behaviour modelling. By having a shared model of what constitutes a plan, process or activity, organisational knowledge can be harnessed and used effectively

SPAR has been a contributing source towards the development of the Core Plan Representation (CPR - refer to http://www.teknowledge.com/CPR2/). CPR is a model that expresses information common to many plan, process, and activity models. The goal is to leverage common functionality and facilitate the reuse and sharing of information between a variety of planning and control systems. The CPR embodies a standard that is general enough to cover a spectrum of domains from planning and process management to workflow and activity models. The representation supports complex, hierarchical plan structures. The initial application of the CPR is in addressing plan interchange requirements of several military planning systems, but the model goes beyond military planning and presents a more general plan representation.

Another effort for defining a planning ontology is is the Process Specification Language (PSL), a neutral representation for manufacturing processes developed by NIST (US National Institute of Standards and Technology - refer to http://ats.nist.gov/psl/). Process data is used throughout the life cycle of a product, from early indications of manufacturing process flagged during design, through process planning, validation, production scheduling and control. In addition, the notion of process also underlies the entire manufacturing cycle, coordinating the workflow within engineering and shop floor manufacturing. The goal of PSL is to create a process interchange language that is common to all manufacturing applications, generic enough to be decoupled from any given application, and robust enough to be able to represent the necessary process information for any given application. This representation would facilitate communication among the various applications because they would all have a common understanding of concepts to be shared.

PLANET is an ontology developed at Information Sciences Institure (ISI) by Yolanda Gil and Jim Blythe, within the DARPA High Performance Knowledge Bases (HPKB) program [31]. PLANET is a reusable ontology for representing plans that is designed to accomodate a diverse range of real-world plans, both manually and automatically created. PLANET makes the following representational commitments to provide broad coverage: First, planning contexts that refer to domain information and constraints that form the backgroud of a planning problem are represented explicitly. The same happens for planning problems and relevant alternative plans for each problem. Second, PLANET maintains an explicit distinction between external constraints (e.g. user advice or preferences) and commitments, which the planning agent elects to add as a partial specification of a plan.

The Enterprise Ontology [84] has been developed by the Artificial Intelligence Applications Institute

at the University of Edinburgh, within the Enterprise Project, the UK government's major initiative to promote the use of knowledge-based systems in enterprise modelling, aiming to support organisations effectively in the Management of Change (refer to http://www.aiai.ed.ac.uk/project/enterprise/). The Enterprise Ontology is a collection of terms and definitions relevant to business enterprises, which are based on the definition of Activities, Organizations, Strategies, Marketing and Time. There are also some additional efforts to define planning ontologies, as for example the Planning Ontology Construction Group (POCG) of the ARPA/Rome Laboratory Planning Initiative (ARPI) or the Planning Ontology Project by John Doyle (http://www.kr.org/doyle/); however additional information for these works is difficult to find.

The numerous independent efforts to develop ontologies for planning applications reveal the importance and the difficulty of this ambitious goal. PDDL has established a widely accepted syntactic framework for defining planning domains and problems, however its lack of semantic annotation makes it difficult for creating/merging/using ontologies. PDDL extensions like the Web-PDDL or the DAML family seem to constitute the natural next step towards a clear, unambiguous, expressive and decidable planning language.

3.3.1 Actions for the Planning Community

Based on the presentation and the discussion in the preceding sections we could identify the following actions that the planning community should consider:

- Active participation in the standarization process of semantically rich languages, in order to ensure that they will support all the planning-related necessary information, as e.g. invariants, control knowledge etc. Existing languages, such as PDDL, should either be extended, or replaced by the new standards.
- Development and standarization of ontologies, both for the Semantic-Web domain and for the traditional ones.
- Support to semantically rich representation languages by the modern planning systems.
- Focus to the application of the planning technology to the Semantic web domain, either for information extraction, or for task accomplishment.

3.4 A KE Support Environment

After having discussed the nature of KE, and the kinds of tools likely to be found in a KE environment, we now use Figure 1 to show the kind of architecture that integrates these artifacts. This "idealised planning KE environment" was inspired by the Planform project proposal [1], although of course it could be changed to other topologies. It may be, for example, that the interface tools to the domain model could be assumed to be interfacing to all aspects of the system, in which case "Planning Application" would be at the core of the environment.

Note that we concentrate here on the knowledge-based aspects of the environment; as the application of planning technology will generally be a complex task, it seems necessary that issues of configuration management, version control etc must also be addressed. These and other vital concerns relating specifically to software engineering and project management will not be considered further, as they lie outside the "knowledge-based" concerns.

Users, Managers and Domain Engineers may want to add and adjust knowledge, apply measurements to the model, inspect the model, animate the model (using a simple planner) or explore the truth of properties of the model.

Existing data and models may form a substantial part of the planning system, and will need a customized acquisition tool to convert it into the internal format.

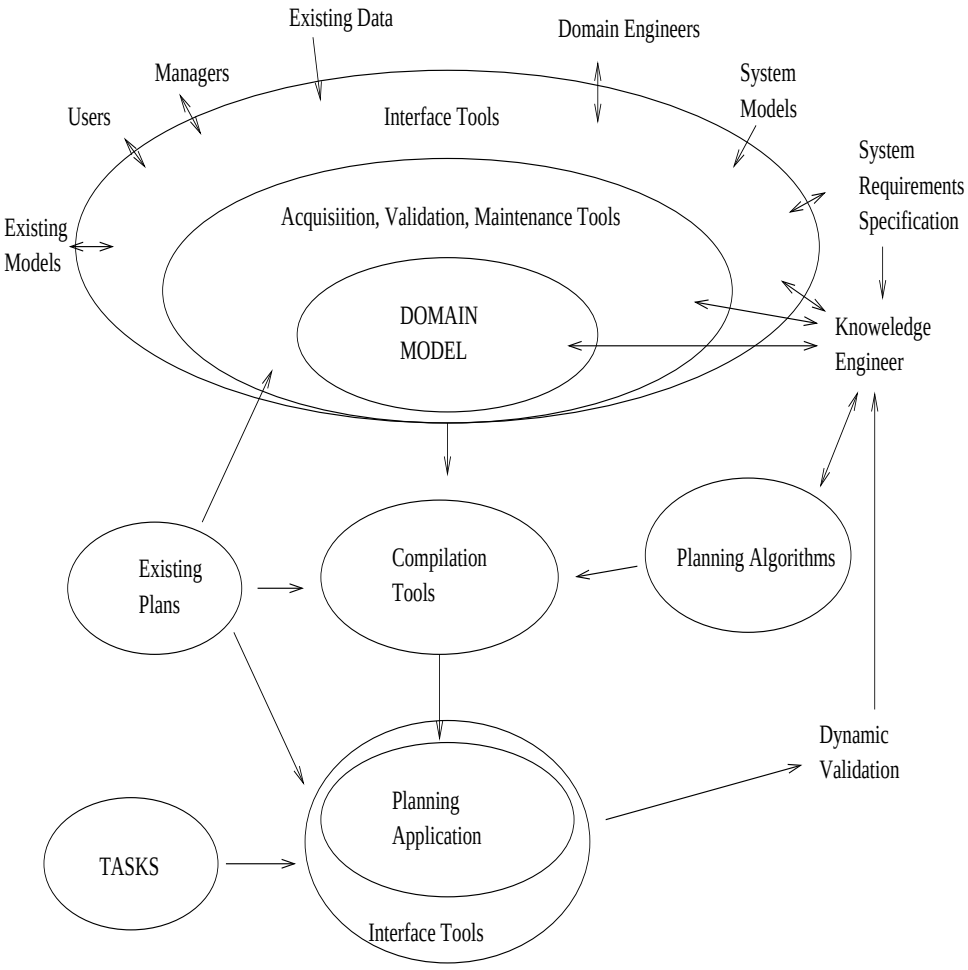

Figure 1: An Idealised Planning Knowledge Engineering Environment

The results of previous analysis and feasibility studies may prove useful - hence information from known system models and from the project's requirements specification will influence the development of the planner.

Existing plans are a database of the kind of plans expected from the final planner. Hence they can be used in may ways: For initial knowledge acquisition, as a source of heuristics for the planner, and as validation of the final planner.

So that knowledge is not "hand coded" into the planning architecture, and to avoid problems in maintenance of the system, it seems sensible to keep separate algorithmic, heuristic and fundamental domain knowledge until these can be fused into a planning application. Ideally, this fusion will be performed by a compilation tool. This tool would, using aspects of the domain, evaluate to find out which planning technology was most appropriate and from this build up a final planning application.

After an initial knowledge acquisition and static validation phase, the compiler can be used to produce an application on which traditional dynamic testing can begin.

4 Related Areas

In this section we consider research areas which contribute to the subject area. We discuss each from a 'Planning' perspective, then make some key points on the way they relate to knowledge engineering in planning.

4.1 Knowledge Based Systems

In the last 20 yeaars or so there has been a paradigm shift in the field of KBS. This was the transformation from the old idea of 'knowledge transfer', where constructing a KBS amounted to extracting the knowledge from experts and encoding it within an expert system 'shell'. This shift re-focussed the field onto 'domain modelling' - with the consequent emphasis on the buiding of a deep causal model prior to an operatational system. The domain model has to embody not just the procedural expert knowledge but the environment in which this knowledge is to be utilised. Several 'modelling frameworks' have been developed (e.g. CommonKads [88] or for more recent work see reference [13]). These support the process of model acquisition and validation, and are underpinned by an overall method of development. CommonKads in particular advocates the use of a series of models during domain capture, each dealing with different aspects of the domain.

An underlying aspiration of recent KBS research is that of *knowledge sharing and re-use*, built on the assumption that it is neither effective nor efficient to consider building a KBS from scratch. The 'knowledge level' view has become prevalent in the literature [68] and emphasises the idea of implementation-independent, and hence more re-usable knowledge bases. Many aspects of KBS are common, in particular common sense knowledge and common strategies for problem solving. KBS researchers have concentrated on several aspects of this need:

reusing/sharing of procedural knowledge: the formulation and re-use of generic problem solving strategies and knowledge bases lies at the heart of the solution to the 'knowledge acquisition bottleneck' in KBS. Problem Solving methods (PSMs) were developed, encouraging the growth of catalogues of 'generic methods'.

reusing/sharing of declarative knowledge: The emergence of standard interfaces and language conventions for KBS is leading to the possibility of standardisation and interoperability.

Another recent development has been the movement in the KBS community to make domain modelling more of a rigorous exercise [85]. Capturing a domain in a model is not just about capturing the 'dynamics': the validity of descriptions (properties/relations) of objects is of crucial significance. This equates to validating that the 'possible worlds' in a model co-incides with actual possible worlds in the domain, as far as possible. This is a very complex KE issue often overlooked in the development of knowledge-sparse

models. Attempting to address the validation issue requires separating out the validation process using the *structure* of a domain modelling language. The accepted wisdom from the field of Formal Methods is to capture the structure, and in particular the *invariants* of a domain in a formal language. A state that satisfies these structural restriction is called 'valid'. Operations are then captured (often using a similar notation to planning). A key verification task is then to check that the initial state observes the invariants, and that if any operation is executing in a valid state the output of the operation is valid. This precipitates an inductive proof that all generated states are valid. It has the benefits that (a) it provides documentation for the developers (b) it helps elliminate errors early development (c) it helps developers analyse the model in greater depth, and helps to reflect the domain being modelled.

4.1.1 Issues for the Planning Community

In planning, the development of application ontologies and planning ontologies is leading to the possibility of planning component standardisation and interoperability. Efforts to develop an ontology for plans can facilitate the interoperability of plan manipulation tools (e.g. using the PLANET ontology [31] or SPAR [80]).

What is required is a review of the KBS knowledge engineering literature, and the creation of a catalogue of tools and techniques that may be relevant to the AI Planning area. Important questions that such a review should answer include: Are these tools actually in use outside the laboratory? How have these tools been evaluated? What lessons can the Planning community learn from the methodology for research and development in related areas?

4.2 Semantic Web and AI Planning

4.2.1 Introduction

Traditionally, AI planning has focused mainly on specialized, non every-day life applications, such as robot navigation, aerospace applications, logistics management, workflow management etc. Although the benefits from the application of AI planning technology in these areas were of great importance, the situation has great prospects to change in the next few years, due mainly to two reasons:

- the wider acceptance of e-commerce as the main way to purchase goods, and
- the gradual transformation of the traditional Web to the richer Semantic Web.

W3C, the World Wide Web Consortium (http://www.w3.org/) defines the Semantic Web as: *"...the abstract representation of data on the World Wide Web, based on the RDF standards and other standards to be defined."* (http://www.w3.org/2001/sw/). A parallel definition is given by Lee, Hendler and Lassila [9]: *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."*

The Semantic Web constitutes a new application domain for planning technology, having the dynamics of both to advance this technology and simultaneously to increase its recognizability and acceptability by the general public. This will lead to an increased demand for high quality planning and scheduling services, which will lead to increased financing of the planning and scheduling research, thus recurrently boosting the progress in this discipline.

The planning problem in the context of the Semantic Web usually has the following form: Given:

- a set of Web Services, i.e. agents on the Web that provide services (usually information gathering or on-line transactions) to other agents, by exposing an interface containing actions definitions, and
- a goal to be achieved, like e.g. get specific information or achieving a specific arrangement,

find a partial ordered set of actions, i.e. interactions with web services, which ensures the achievement of the imposed goal.

There are several technical problems that arise in the context of Semantic Web. These concern both knowledge representation and planning algorithms. Concerning knowledge representation, there is an imperative need for a widely-accepted web-services desctiption language, in order to be feasible for a planning agent to exploit the full range of available resources in the web. Having the experience of the boost that the lisp-like Planning Domain Definition Language (PDDL, [30]), as well as the international planning competitions (in the conferences AIPS-98, AIPS-00 and AIPS-02) gave to the planning research in traditional domains during the last years, it is easy to predict how valuable a widely accepted language for describing Semantic Web domains would be.

However, there is a major difference between web-enabled languages and PDDL. PDDL is a syntactic language, which only defines the way operators and states have to be written. There is no special care for the atomic terms that are used to name predicates and actions. Actually, PDDL lacks its semantic counterpart. However, this was not a problem, since in most domains where PDDL has been used, the closed world assumption is supposed. Even in cases where an open world is assumed, this only concerns the number of the objects.

On the other hand, the Semantic Web is an open world. It is not the number of objects (i.e. agents) that is practically infinite, but also the number of concepts which continuously increases. New web services arise, providing for new actions that have to be absorbed and used by the planning agents. So, in order for a planning agent to absorb the new actions, understand their meaning, their prerequisites and their effects, a common vocabulary has to be defined, giving conceptual meaning in each atomic term and establishing conceptual relations among them. Usually, this is achieved with the use of ontologies, i.e. concept hierarchies, which are attached to Web documents thus giving to them semantic meaning. Of course, we are still too far from the adoption of the One Global Ontology, which will cover every aspect of our knowledge, however numerous ontologies have established, accepted and used by specific target communities, whereas preliminary attempts for automatic or semiautomatic correspondence or merging of ontologies have been undertaken.

The open world assumption also affects most of the modern planners. Actually, most of the planners that work by exhaustively enumerating all the ground facts and actions (e.g. heuristic and Graphplan/Satplan planners) have to adapt to the Semantic Web domain. Moreover, the need for parallel plans, optimized with respect to several criteria (including response time and overall cost) is of great importance, in order for the planning technology to be adopted by the general public.

In the following we review the current status in defining languages for describing web services and the attempts to construct planners that work on the web.

4.2.2 Web Languages

The Semantic Web differs from the traditional Web in that data are accompanied not only by formating annotation (as in the case of HTML) but also by conceptual characterizations. So, web-based agents have the ability to understand, reason and finally plan for their own goals. Semantic Web languages include, among others, XML, RDF, RDFS (information on these is available at http://www.w3.org/), SHOE (http://www.cs.umd.edu/projects/plus/SHOE/) and DAML+OIL [62].

The XML (Extensible Markup Language) was originally designed for large-scale electronic publishing; however it plays an important role in data exchange over the Web. XML aims mainly at storing, carrying and exchanging data. It is characterized by a very simple but strict syntax, which facilitates the creation of software that manipulate XML documents. Moreover, XML Namespaces provide a means of semantic characterization of the information stored in XML documents, by identifying element and attribute names and solving name conflicts.

The RDF (Resource Description Framework) complements XML in that it provides a means for cataloguing data in the web for effective and efficient retrieval. RDF files use triplets of the form <Resource, Property, Value> to characterize web resources, usually XML documents, with specific properties.

RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties may also represent relationships between resources. RDF however, provides no mechanisms for describing these properties, nor does it provide any mechanisms for describing the relationships between these properties and other resources. That is the role of the RDF vocabulary description language, RDF Schema (RDFS). RDF Schema defines classes and properties that may be used to describe classes, properties and other resources. So, RDFS is recognisable as an ontology language. The combination of RDF/RDFS allows applications to be programmed with a standard method to process metadata and exchange information without worrying about interoperability.

However, RDFS is not a suitable foundation for the Semantic Web, since it is too weak to describe resources in sufficient detail. Three inefficiencies of the RDF/RDFS combination are the following ones:

- they are not formally specified
- they do not have adequate expressive power
- they do not provide automated reasoning support

The above inefficiencies restrict the chances of automated planning systems to fully exploit the opportunities of the Semantic Web. Several languages have developed to meet the above requirements. Some of them are mentioned in the following paragraphs.

SHOE (http://www.cs.umd.edu/projects/plus/SHOE/) is a simple HTML Ontology extension, which allows web page authors to annotate their web documents with machine-readable knowledge. SHOE claims to make real intelligent agent software on the web possible. SHOE is no longer extended, since SHOE researchers have been targeted into OWL and DAML+OIL web ontology languages.

OIL (Ontology Interface Layer, refer to http://www.ontoknowledge.org/oil/) is a joint standard for specifying and exchanging ontologies, developed within the OntoKnowledge Project (http://www.ontoknowledge.org/). It is a proposal for a web-based representation and inference layer for ontologies, which combines the widely used modeling primitives from frame-based languages with the formal semantics and reasoning services provided by description logics. It is compatible with RDF Schema (RDFS), and includes a precise semantics for describing term meanings (and thus also for describing implied information).

The DAML language (DARPA Agent Markup Language - refer to http://www.daml.org/) is being developed as an extension to XML and the Resource Description Framework (RDF), as an effort to focus on the eventual creation of a web logic language. DAML-ONT (http://www.daml.org/2000/10/daml-ont.html), released in 2000, was the initial ontology core of DAML, roughly corresponding to a frame-based / description logic starting place, allowing the definition of classes and subclasses, their properties, and a set of restrictions thereon.

DAML and OIL efforts were merged to produce DAML+OIL language (http://www.daml.org/2001/03/daml+oil-index.html) a semantic markup language for Web resources. It builds on earlier W3C standards such as RDF and RDF Schema, and extends these languages with richer modelling primitives. DAML+OIL provides modelling primitives commonly found in frame-based languages and values from XML Schema datatypes. The language has a clean and well defined semantics. DAML+OIL is going to become a W3C standard, under the new name OWL (Web Ontology Language - refer to
http://www.w3.org/TR/webont-req/).

Of special interest for the planning technology is the description of web-services within the ontologies. Such services, like e.g. booking a hotel room or ordering a meal, constitute executable actions, the counterpart of the information gathering actions. DAML-S (S stands for Services - refer to http://www.daml.org/services/) is a markup language for describing Web services, jointly developed by ISI, BBN, CMU, Nokia, SRI and Yale. DAML-S builds on top of DAML and supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-intepretable form. DAML-S markup of Web services

intends to facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring. DAML-S is supported by an ontology editor (http://www.ksl.stanford.edu/projects/daml/), with a graphical and form-entry user interface, which allows for defining composite services based on simpler ones. Finally, a prototype tool for automated synthesis of composite web services has been implemented.

A parallel approach for defining a web service ontology is WSDL (http://www.w3.org/TR/wsdl), submitted to W3C in 2001. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

A very interesting effort, aiming to provide a test-bed for experminets with intelligent agents and web-services is AgentCities (http://www.agentcities.org/). It is about a worldwide initiative designed to help realise the commercial and research potential of agent based applications by constructing a worldwide, open network of platforms hosting diverse agent based services. The ultimate aim is to enable the dynamic, intelligent and autonomous composition of services to achieve user and business goals, thereby creating compound services to address changing needs. The initiative aims to build on several innovative technologies including agent technology, Semantic Web technologies, UDDI discovery services, eBusiness standards and Grid Computing. Potential application areas range from eHealth and eLearning to manufacturing control, digital libraries, travel and entertainment services.

4.2.3 Issues for the Planning Community

The Semantic Web domain adds another agent in the KE team: that of the website developer. Now the development of a site needs to take into account not only the human-computer interface, but also the agent-to-agent one. In other words, the website developer has to create two (usually identical) interfaces, the second one of them descibing the services exposed by the site, using one of the languages mentioned in the previous paragraphs.

The questions that arise from the above presentation, with respect to the standard AI planning domain languages such as PDDL, are obvious: Is there any need for another language describing web information and web services? Is it worth extending languages like PDDL in order to cover web applications? Or are existing languages, like DAML, OWL or WSDL, sufficient for planning applications? The answer seems to be somewhere in the middle.

Drew McDermott introduced Web-PDDL, a Web version of PDDL which supports XML namespaces and flexible notations for axioms. Web-PDDL uses Lisp-like syntax and is a strongly typed first-order logic language. Moreover, a translator between Web-PDDL and DAML, named PDDAML, is provided (http://cs-www.cs.yale.edu/homes/dvm/daml/). Similar translators between DAML-S and PDDL are provided (http://www.ksl.stanford.edu/projects/daml/). These works show that there is no great semantic difference between traditional AI planning languages like PDDL and new web-service languages, so it is possible for these two areas to converge.

4.3 Formal Methods in Software Engineering

Formal Methods in software engineering covers (1) the capture and analysis of a (formal) specification of software within a structured formal language, and (2) the refinement of a formal specification into an efficient implementation. The formal specification language has to be appropriate to the application at hand, be sufficiently abstract to allow formal reasoning, and be well supported with a tools environment. The primary concerns in formal methods is to show that the initial specification is internally consistent and externally valid, and to prove that the derived implementation is correct with respect to the specification. These processes are meant to improve the quality of the software process and product, as they are

aimed at the early identification and removal of bugs. Superficially at least, there is a strong similarity between formal specification languages and planning languages. Both kinds of languages are designed to allow engineers represent actions precisely and declaratively. Take VDM-SL (the Vienna Development Method's Specification Language [43]) and a STRIPS-language. Both are based around the notion of a state, allow the developer to create operators, and in both cases those operators are defined using pre- and postconditions. Further, they are both based on the assumptions of closed world, default persistence and instantaneous operator execution. VDM encourages the creation of state invariants for validity and documentation purposes; state invariants are also used in some planning languages, but the main rationale here seems to be plan generation speed-up.

The difference between those using formal specification to describe systems and those using a planning language to model a planning domain, is that in the former case the specification is used as a blueprint for design, whereas in the latter case the specification is used as input to a planner to be reasoned with in order to construct plans to achieve goals. States in languages such as VDM-SL and Z [72] are built up from mathematical data types such as sets, mappings, sequences etc. With the exception of work in deductive planning [10], much of the work carried out in planning research assumes little or no structure to types (predicates are often assumed to be "function free").

For an introduction to the algebraic and model-based formal specification one can consult reference [83]. This reference also describes how a Tweak-like planner can be be represented in VDM-SL, how its plan-space operations can be specified using VDM operations, and how the whole specification can be rigorously translated into a logic program.

4.3.1 Issues for the Planning Community

We can learn from the development of formal methods. One particularly successful use of a formal system in AI planning and scheduling is in the use of Petri Nets (e.g. [63]). It is agreed that, in general, the take up of Formal Methods in mainstream software development has been problematic, mainly for the reason that the formalisms are not understandable to most stakeholders in the system being developed. In the same way, we would not want to let our users look at pieces of ADL or scrutinize HTN operators! One of the approaches being pursued in formal methods is the use of "methods integration", that is using an informal, graphical front-end method to allow non-mathematicians to develop the specification. Languages such as Z and methods such as OMT have been fused this way [5]. After initial informal development, tools translate the diagrammatic language to a formal specification language. A formalist then fills in more details of the specification (the informal language invariably leads to underspecified action schemata) and the result is (ideally) mapped back to the friendly front end for further validation. This kind of methods integration may well help open up the use of formal planning domain descriptions languages such as PDDL to more widespread use and acceptance.

4.4 Machine Learning

4.4.1 Introduction

In the AI research literature, there are many systems that use "learning" tools in the development of a planner, or within the planning product itself. This is an attractive area from the symbolic machine learning (ML) point of view, as planning involves the acquisition of knowledge and high level cognitive skills. Planning programs can then be used as the performance components in evaluating learning techniques. ML techniques, on the other hand, can help planning in several ways. Here, we will only give a short summary. A comprehensive state of the art can be found in [90].

- **Knowledge acquisition:**

  ML techniques could be used to remove the bottleneck of eliciting knowledge in much the same way as ML techniques have been used in front-ends for expert systems. For example, it may be more useful to induce the symbolic specification of complex actions than trying to get an expert

to describe them in formal terms. For instance, in references [14,28] experimentation is used in the domain to extract domain knowledge, while in [68] a domain expert agent uses observation with the same purpose, and in [24] domain actions are learned from an autonomous agent moving around in a robotic domain. More recently, similar techniques have been applied to learning methods for HTN planners and tested in a simplified NEO domain [42]. A recent trend is to combine ML and interaction with the user/expert, within a KE graphical environment [61, 25].

- **Model validation:**

  Complex models can be incorrect or incomplete. If classified training data is available then, for example, a theory revision tool could be used to help identify and remove errors [60, 61, 64].

- **Improving the quality of plans:**

  Finding plans of good or optimal quality is very important and it is known that this is more complex than just finding valid plans. Here "quality" is any metric applied to a plan: number of operators in the plan, economic cost of executing the plan, time required to execute it, , etc. Examples of tools that learn knowledge to augment particular planners so that they output quality plans are QUALITY [54], Hamlet [13], and Scope [19]. Instead of learning control knowledge, other researchers learn rewriting rules to transform low quality plans into high quality ones [7].

- **Improving planning efficiency:**

  It is well known that solving even medium sized planning problems is a hard task for domain independent planning. A well researched way of improving planner efficiency is by using ML to learn control knowledge. This will be considered in the next section.

4.4.2 Exploiting ML for Planning Speed-up

Perhaps the most popular application of ML to planning is in plan generation speed-up, where learning concerns itself with improving planning efficiency (decreasing resources in time and space). Acquiring heuristics for any knowledge-based task is as difficult as acquiring the model's dynamics and structure. In the case of planning, to hand-code heuristics the user would need to know how the planning engine works to be able to define correctly those heuristics. Therefore, ML techniques have the potential to overcome this knowledge acquisition bottleneck by automatically learning those heuristics, with the possibility of presenting them to the user to refine or validate them. There have been many approaches, for example:

- **Macro-operator learning:** Macro-operators are sequences of planning operators that have been compiled into a form that can be easily retrieved to form a partial solution to a planning problem. Early work concentrated on macros to help in general problem solving [41,56]. In the FM system [48] macros were used to make problem solving in STRIPS-robots domains highly efficient.
- **Analogy/Case Based Reasoning:** This approach uses planning problems that have already been solved to guide the solution of similar problems Chef [30], Analogy [67], CAPlan/CbC [51,7].
- **Learning sequences of subgoals:** It is also possible to learn sequences of subgoals as stepping stones in the planning process. For example SteppingStone [60], EAS [61] (which is also CBR based).
- **Learning heuristics:** Planning is usually seen as a search process, therefore it is possible to use heuristics to guide this process. Techniques belong to three main categories:
  - **Trace based "deductive" systems** are similar to DA techniques in that they use the domain description, but in addition rely on a few traces of the planner after solving planning problems to create control knowledge for that domain. Most of them use EBL Prodigy-EBL [49], ULS [16], for total order planners [8], and SNLP+EBL [37] for partial order planners. There are also hybrid systems, like DERSNLP+EBL [34] that combines CBR and EBL. FM [47] used a deductive technique to create the hypothesis space for the conditions of a heuristic that was further refined using induction as more traces became available.

- **Trace-based systems can be purely inductive**, relying mostly on planning traces to build control knowledge. They generalize from the specific traces to build planning control knowledge, examples are given in references [49,21,43,42].
- **Multi-strategy systems:**. Inductive systems require many examples and if they are incremental, depend on the ordering of learning instances [38]. On the other hand, deductive systems need completely correct theories, usually produce knowledge that is too specific, and are prone to the utility problem. Multi-strategy systems combine deductive and inductive approaches to overcome these limitations (FM [46], AxA-EBL [17], Hamlet [13], SCOPE [19], EvoCK [3]).

- **Learning policies:** Instead of learning control knowledge for a planner, some researchers learn policies, that determine which operator should be applied at any given planning situation (this is reminiscent of universal planning) [44, 56].
- **Learning constraints:** nowadays, a new generation of very fast planning systems based on propositional representations/constraint satisfaction have come to the fore. It is more difficult to improve the efficiency of these planners, but there are already some results, like [41], that learn declarative rules which are added to the planning problem as additional constraints.

4.4.3 Issues for the Planning Community

ML is as relevant for KE in planning as it is for KE in other fields: ML-based tools have the potential to reduce the knowledge acquisition bottleneck. Taking into account what has already been achieved, we believe that the focus of future research should encompass the following areas:

- **Real-world domains:** it could be said that current research on ML applied to planning has shown that ML techniques are useful, but they have only been tested in simple domains. Although some studies show that the knowledge learned scales to hard instances in those domains (e.g. 20 blocks in the blocks world), future studies should consider medium-sized instances in real-world domains.
- **Improving the quality of plans:** although this is a very important issue, not much work has been done in this area. Plan quality improvement should become a focus for research in the immediate future. This is likely to be a tough task, because finding a solution in some domains is "easy", whereas finding good or optimal solutions is very hard. Further complications will arise when multiobjective quality measures (like time and cost) are considered.
- **User interaction:** and yet, it has been shown that the learned knowledge still contains errors that could be corrected by a human [4]. Therefore, future studies should consider putting the human back into the learning cycle, possibly assisted by graphical tools, to validate, correct, and improve learned knowledge.
- **Integration of ML into KE environments:** A lot has to be done in studying the interaction between a ML system and a planning expert/user/knowledge engineer (for instance, to validate and correct knowledge automatically acquired). Integrating ML into KE graphical tools should be of primary importance if ML is to be useful to Planning KE community.
- **ML for modern planners:** Nowadays, there is a new generation of fast planners based on propositionalization, constraint satisfaction, and heuristic search. However, most of the work done in speed-up planning, and learning in general concerns older planning systems. New work should be done to apply previous learning methods and to develop new ones for these planners. Although they are already very fast, domain independent planning will always be a NP task, and domain dependent knowledge will be needed to solve hard instances in tough domains. Some care will be required so as not to damage performance in small and medium-sized problems because of the utility problem. In any case, planners becoming more and more efficient should make improving plan quality occupy the center stage in future research. Also, other features that go beyond STRIPS, like duration, numerical quantities, scheduling,

uncertainty, hierarchical task network planning (HTN), ... should also be addressed from the control knowledge perspective.

- **Techniques and representation of control knowledge:** It is difficult to determine which techniques should be studied in the future, but we believe that multi-strategy learning has good chances of improving results further, by combining the appropriate standalone ML techniques. Also, researchers should study whether current languages for representing heuristics are powerful enough to achieve the desired goals (i.e. achieving the maximum efficiency or quality possible in a particular domain). Probably, more powerful languages will be necessary, including the ability to carry out complex computations and handling numeric quantities. Of course, the more powerful, the more difficult it will be to automatically learn heuristics for those languages.

5 Roadmap Summary

5.1 Summary of Problems

The roadmap points out some general problems to be overcome. These include:

1. Little knowledge of the creation, validation and maintenance of large domain models exists in the planning community. There is a general lack of experience (especially in Europe) of knowledge engineering concepts and methods in this area, and we have much to learn and techniques to import from related knowledge engineering work.
2. The evaluation of KE tools and methods is problematic, and tends to be harder than the evaluation of, say, a planning algorithm. This is seen as a barrier to future research as researchers find it harder to publish in the planning literature.
3. We need to learn how to characterize domains, and hence build up knowledge about how to match up planning technology with domain models. An example of work in this direction, based on statistical analysis, may be found in [40].
4. KE involves a group of stakeholders with differing backgrounds and knowledge of computing. In particular, it includes consideration of Human Factors.
5. Planning research has a history of association with toy problems where KE issues are not relevant. There has been a focus in the planning community on theoretical aspects of techniques that do not scale up to real-world planning problems.

5.2 Summary of Actions

The problems lead to some general actions that the community should carry out. These are summarized in Figure 2.

1. Encourage the planning research community to develop greater appreciation for the applied end of the research continuum, as argued in reference [89].
2. Survey the areas of knowledge acquisition, machine learning, requirements engineering and formal methods in software engineering for tools, techniques, and methods that may be relevant to planning.
3. Distill experience and induce general methods from the experience of previous applications of planning technology.

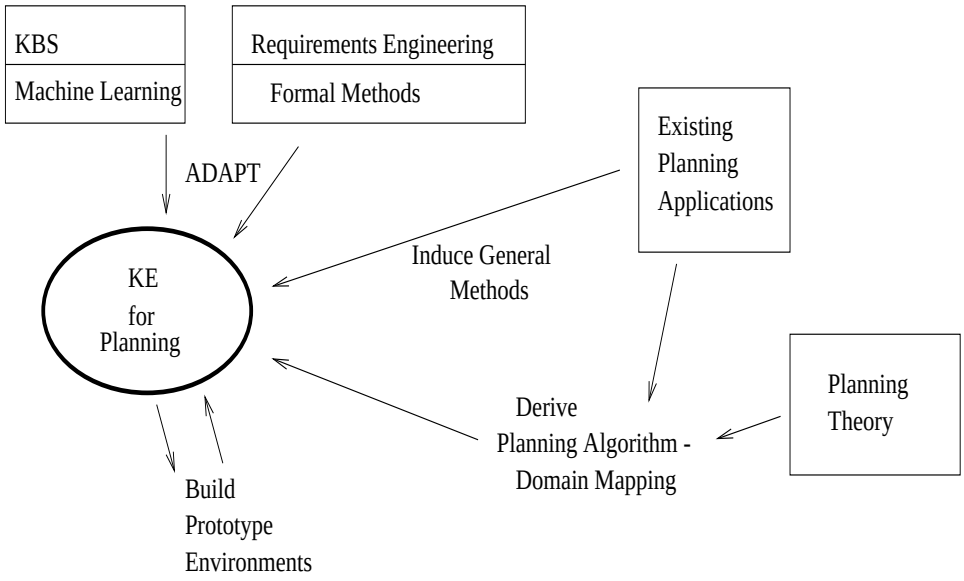

Figure 2: Roadmap Summary

4. Attempt to build basic, integrated engineering environments for building planning applications. This may also involve promoting standards and/or formalisms that enable integration of planning engines and support tools developed separately.

5. Find a research methodology that allows efficient evaluation of work in KE to allow a route to publication and hence attract more research effort.

6. Build on previous work to create a planner taxonomy usable by knowledge engineers.

7. Develop a classification system and/or vocabulary for describing the characteristics of domains (as advised in reference [39])

5.3 Footnote: Knowledge Engineering and PLANET

An attempt to summarize the real world problem areas dealt with in PLANET in terms of Knowledge Engineering would be worthwhile. For example, in WorkFlow Management there are two key contributions from KE in AI planning. First, existing workflow systems take a model of a business process and then use that to co-ordinate the execution of an instance. If one views a business process description as a HTN operator and a business process instance as a instantiation of such an operator (i. e. a plan) then the KE work can help as follows: We have worked out ways of representing time, resource, and state knowledge. If we then develop methods for modeling this knowledge then these should help business process modeling too. This kind of modeling may well be of use to standards bodies or commercial consortia such as the Workflow Management Coalition. Second, at the moment business process descriptions are fixed (universal plans). AI planning can help in synthesizing a process instance that takes into account the situation and purpose of its execution. If we had a good method for capturing process knowledge and for reasoning with it then the workflow community would (a) use such a method and (b) provide a market for the actual use of AI planning technology. From the workflow TCU we can learn about the requirements of action representation. An example from document versioning: Basically this requires a planner to handle the creation of new objects which is difficult with current planning technology - especially when quantification is required. However, this is a straightforward everyday requirement.

References

[1] Planform: An open environment for building planners. http://scom.hud.ac.uk/planform.

[2] M. Aben, J. Balder, and F. van Harmelen. Support for the formalisation and validation of KADS expertise models. Technical Report KADS-II/M2/UvA/DM2.6a/1.0, ESPRIT, 1994.

[3] R. Aler, D. Borrajo, and P. Isasi. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence Journal*, 141(1-2):29–56, October 2002.

[4] Ricardo Aler and Daniel Borrajo. On control knowledge acquisition by exploiting human-computer interaction. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pages 141–149, Toulouse (France), April 2002. AAAI Press.

[5] P. M. Allen and L. T. Semmens. Integration of structured methods and formal notations. In *International Conference on Information Systems Development*, Bled, Slovenia, September 1994.

[6] S. Amarel. On representations of problems of reasoning about actions. In D. Michie, editor, *Machine Intelligence 3*, pages 131–171. Edinburgh University Press, Edinburgh, 1968.

[7] J.L. Ambite, C.A. Knoblock, and S. Minton. Learning plan rewriting rules. In *Proceedings of the Fifth Intenational Conference on Artificial Intelligence Planning and Schedulling*, Breckinridge. CO (USA), April 2000.

[8] A. Barr and E. A. Feigenbaum, editors. *The Handbook of Artificial Intelligence. Vol. 1.* HeurisTek Press, Stanford, CA and William Kaufmann, Los Altos, CA, 1981.

[9] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 2001.

[10] S. Biundo and W. Stephan. Modeling planning domains systematically. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 1996.

[11] Susanne Biundo and Maria Fox, editors. *Proceedings of the European Conference on Planning.* Springer, September 1999.

[12] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[13] J. Blythe, J. Kim, S. Ramachandran, and Y. Gil. An Integrated Environment for Knowledge Acquisition. In *Proceedings of the Interenational Conference on User Interfaces*, 2001.

[14] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *[11]*, pages 359–371, 1999.

[15] A. Cesta and A. Oddi. DDL.1: A formal description of a constraint representation language for physical domains. In *[29]*. 1996.

[16] J. M. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry breaking predicates for search problems. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 1996.

[17] M. Davidson and M. Garagnani. Preprocessing planning domains containing Language Axioms. In *Proceedings of the Twentyfirst Workshop of the UK Planning and Scheduling SIG (PlanSIG-02)*, pages 23–34, Delft, NL, 2002.

[18] F.L. Dretske. *Knowledge and the Flow of Information.* MIT Press, Cambridge, MA, 1981.

[19] Oren Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):265–301, 1993.

[20] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research – Special issue on the 3rd International Planning Competition*, 2003. (forthcoming).

[21] Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of AI Research*, 9:367–421, 1998.

[22] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning domains. Technical Report tr 1/99, Durham University, 1999.

[23] M. Garagnani. A correct algorithm for efficient planning with preprocessed domain axioms. In M. Bramer, A. Preece, and F. Coenen, editors, *Research and Development in Intelligent Systems XVII: Proceedings of ES2000*, pages 363–374, Cambridge, England, 2000. Springer-Verlag.

[24] M. Garagnani. Model-Based Planning in Physical domains using SetGraphs. In M. Bramer, A. Preece, and F. Coenen, editors, *Proceedings of AI-2003*, Cambridge, England, December 2003. Springer-Verlag. (to appear).

[25] Garland, Tyall, and Rich. Learning hierarchical task models by defining and refining examples. In *Proceedings of the First International Conference on Knowledge Capture*, pages 44–51, 2001.

[26] B.C. Gazen and C.A. Knoblock. Combining the expressivity of UCPOP with the Efficiency of Graphplan. In *Proceedings of the Fourth European COnference on Planning (ECP-97)*, pages 221–233, Toulouse, France, 1997.

[27] M.P. Georgeff. Planning. *Annual Review of Computer Science*, 2:359–400, 1987.

[28] Alfonso Gerevini and Lenhard Schubert. Inferring state constraints for domain-independent planning. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 905–912, 1998.

[29] M. Ghallab and A. Milani, editors. *New Directions in AI Planning.* IOS Press (Amsterdam), 1996. Proceedings of the 3rd European Workshop on Planning (EWSP95), Assisi, Italy, September 27-29, 1995).

[30] Malik Ghallab, Adele Howe, Craig A. Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wikins. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computitational Vision and Control, 1998.

[31] Y. Gil and J. Blythe. PLANET: a shareable and reusable ontology for representing pla ns. In *Proceedings 17th International Conference on AI*, 2000.

[32] J. Glasgow, N.H. Narayanan, and B. Chandrasekaran, editors. *Diagrammatic Reasoning.* MIT Press, Cambridge, MA, 1995.

[33] S. Greenspan, J. Mylopoulos, and A. Borgida. On formal requirements modeling languages: RML revisited. In *International Conference on Software Engineering.* IEEE Computer Science Press, 1994.

[34] Joseph Y. Halpern and Moshe Y. Vardi. Model Checking vs. Theorem Proving: A Manifesto. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knoweldge representation and Reasoning: Proceedings of the 2nd International Conference (KR91)*, pages 325–332, 1991.

[35] E. Hammer. Reasoning with sentences and diagrams. In G. Allwein and J. Barwise, editors, *Working papers on Diagrams and Logic. Preprint IULG-93-24*, pages 120–143, Indiana Univ., Bloomington, IN, 1993.

[36] Patrik Haslum and Peter Jonsson. Planning with reduced operator sets. In Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *Proceedings of the Conference on Artificial Intelligence Planning & Scheduling*, pages 150–158, April 2000.

[37] Patrik Haslum and Ulrich Scholz. Domain knowledge in planning: Representation and use. In Derek Long, Drew McDermott, and Sylvie Thiébaux, editors, *ICAPS'03 Workshop on PDDL*, pages 69–78, May 2003.

[38] P.J. Hayes. Some problems and non-problems in representation theory. In *Proceedings of the AISB Summer Conference*, pages 63–79, University of Sussex, Brighton, England, 1974.

[39] J. Hertzberg. On building a planning toolbox. In *[29]*. 1996.

[40] Adele E. Howe, Eric Dahlman, Christopher Hansen, Michael Scheetz, and Annelise von Mayrhauser. Exploiting competitive planner performance. In *[11]*, pages 60–72, 1999.

[41] Yi-Cheng Huang, Bart Selman, and Henry Kautz. Learning declarative control rules for constraint-based planning. In *Proceedings of the 17th International Conference on Machine Learning*, pages 415–422, Stanford (USA), 2000.

[42] O. Ilghami, H. Muñoz-Avila D. Nau, and D. Aha. Camel: Learning methods for htn planning. In *Proceedings of the Conference on AI Planning and Scheduling (AIPS'02)*, Toulouse (France), 2002.

[43] C. B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, 2 edition, 1990.

[44] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence Journal*, 113:125–148, 1999.

[45] R. Khardon and D. Roth. Reasoning with Models. *Artificial Intelligence*, 87:187–213, 1996.

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

[47] Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.

[48] Z. Kulpa. Diagrammatic representation and reasoning. *Machine GRAPHICS & VISION*, 3(1/2):77–103, 1994.

[49] J.H. Larkin and H.A. Simon. Why a diagram is (sometimes) worth ten thousands words. *Cognitive Science*, 11:65–99, 1987.

[50] H. Levesque. Is reasoning too hard? In *Proceedings of the Third NEC Research Symposium*, 1992.

[51] V. Lifschitz. On the semantics of STRIPS. In M.P. Georgeff and Lansky, editors, *Proceedigns of 1986 Workshop: Reasoning about Actions and Plans*, 1986.

[52] R.K. Lindsay. Images and Inference. In J. Glasgow, N.H. Narayanan, and B. Chandrasekaran, editors, *Diagrammatic Reasoning*, chapter 4, pages 111–135. MIT Press, Cambridge, MA, 1995.

[53] D. Liu and T.L. McCluskey. The OCL Language Manual, Version 1.2. Technical report, Department of Computing and Mathematical Sciences, University of Huddersfield (UK), 2000.

[54] Derek Long and Maria Fox. Automatic synthesis and use of generic types in planning. tr 3/99, Durham University, 1999.

[55] Derek Long and Maria Fox. Extracting route-planning: First steps in automatic problem decomposition. In *AIPS Workshop on Analysing and Exploiting Domain Knowledge for Efficient Planning*, 2000.

[56] M. Martin and H. Geffner. Learning generalized policies in planning using concept languages. In *Proceedings of the 7th Conference on Knowledge Representation and Reasoning*, Colorado (USA), 2000.

[57] J. McCarthy. Programs with common sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes, Vol. 1*, pages 77–84, 1958.

[58] J. McCarthy and P.J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4. Edinburgh University Press, 1969.

[59] T. L. McCluskey. PDDL: A Language with a Purpose? In *Proceedings of the ICAPS-2003 Workshop on PDDL, International Conference on Automated Planning and Scheduling, Trento, Italy*, 2003.

[60] T. L. McCluskey and M.M.West. The automated refinement of a requirements domain theory. *Journal of Automated Software Enginnering. Special Issue on Inductive Programming*, 6(2):195–218, May 2001.

[61] T. L. McCluskey, N. Richardson, and R. Simpson. An interactive method for inducing operator descriptions. In *Proceedings of the 6th International Conference on AI Planning and Scheduling (AIPS-2002)*, Toulouse (France), April 2002.

[62] Deborah L. McGuinness, Richard Fikes, James Hendler, and Lynn Andrea Stein. *IEEE Intelligent Systems*, 17, 2002.

[63] A. R. Moro, H. Yu, and G. Kelleher. Applying new search methodologies for scheduling FMS using petri nets. In *Proceedings of the 17th UK PlanSIG*, University of Huddersfield, UK, September 1998.

[64] M. M.West and T. L. McCluskey. The application of machine learning tools to the validation of an air traffic control domain theory. *International Journal on Artificial Intelligence Tools*, 10(4):613–637, December 2001.

[65] K. Myers and K. Konolige. Reasoning with analogical representations. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*, pages 189–200. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1992.

[66] K.L. Myers and D.E. Wilkins. *The Act-Editor User's Guide: A Manual for Version 2.2.* SRI International Artificial Intelligence Center, Menlo Park, CA, September 1997.

[67] Bernhard Nebel, Yannis Dimopoulos, and Jana Koehler. Ignoring irrelevant facts and operators in plan generation. In *[11]*, pages 338–350, 1997.

[68] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18:87 – 127, 1982.

[69] S.E. Palmer. Fundamental Aspects of Cognitice Representation. In E. Rosch and B.B. Lloyed, editors, *Computing and Categorization*, chapter 9, pages 259–303. Earlbaum, Hillsdale, NJ, 1978.

[70] J.L. Pollock. Perceiving and Reasoning about a Changing World. *Computational Intelligence*, 14(4):498–562, 1998.

[71] S. Polyak. A common process methodology for engineering process domains. In *Proceedings of the Systems Engineering for Business Process Change (SEBPC) workshop, SMBPI: Systems Modelling for Business Process Improvement*, University of Ulster, March 1999.

[72] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z.* Prentice-Hall, 1991.

[73] Ioannis Refanidis and Ioannis Vlahavas. GRT: A domain independent heuristic for STRIPS worlds based on greedy regression tables. In *[11]*, pages 346–358, 1999.

[74] Jussi Rintanen. An iterative algorithm for synthesizing invariants. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2000.

[75] Ulrich Scholz. Action constraints for planning. In *[11]*, pages 148–160, 1999.

[76] M. Shanahan. *Solving the Frame Problem.* MIT Press, Cambridge, MA, 1997.

[77] L. G. Shaw and B. R. Gaines. Requirements acquisition. *Software Engineering Journal*, 11:149–165, 1996.

[78] H.A. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, MA, 1981.

[79] A. Sloman. Afterthoughts on analogical representations. In *Proceedings of the First Workshop on Theoretical Issues in Natural Language Processing (TINLAP-1)*, pages 164–171, Cambridge, MA, 1975.

[80] A. Tate. Roots of SPAR - shared planning and activity representation. *Knowledge Engieering Review*, 13:121 – 128, March 1998.

[81] A. Tate, S. Polyak, and P. Jarvis. TF method: An initial framework for modelling and analysing planning domains. In *AIPS-98 workshop on "Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice"*, Carnegie-Mellon University, June 1998. AAAI Technical Report WS-98-03.

[82] Sylvie Thiébeaux, Jörg Hoffmann, and Bernhard Nebel. In Defense of PDDL Axioms. In *Proceedings of ICAPS'03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*, Trento, Italy, June 2003.

[83] J. G. Turner and T. L. McCluskey. *The Construction of Formal Specifications: An Introduction to the Model-Based and Algebraic Approaches.* McGraw-Hill series in Software Engineering. McGraw-Hill, 1994.

[84] M. Uschold, M. King, S.Moralee, and Y. Zorgios. The Enterprise Ontology. *Knowledge Engieering Review*, 13, March 1998.

[85] F. van Harmelen and D. fENSel. Formal Methods in Knowledge Engineering. Technical report, The University of Amsterdam, 1995.

[86] Klaus Varrentrapp, Ulrich Scholz, and Patrick Duchstein. Design of a testbed for planning systems. In Lee McCluskey, editor, *AIPS'02 Workshop on Knowledge Engeneering Tools and Techniques for AI Planning*, pages 51–58, April 2002.

[87] Dimitris Vrakas, Grigorios Tsoumakas, and Ioannis Vlahavas. Towards adaptive heuristic planning through machine learning. In Tim Grant and Cees Witteveen, editors, *UK Planning and Scheduling SIG Workshop*, pages 12–21, November 2002.

[88] B. J. Wielinga, A. Th. Schrieber, and J. Breuker. KADS - a modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5 – 53, 1992.

[89] D. Wilkins and M. desJardins. A Call for Knowledge-based Planning. In *Procceedings of the 2nd NASA International Workshop on Plan ning adn Scheduling for Space*, page 187, 2000.

[90] Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 2003.