Engineering Knowledge for Automated Planning: Towards a Notion of Quality

Thomas L. McCluskey
University of Huddersfield
Queensgate
Huddersfield, United Kingdom
HD13DH
lee@hud.ac.uk

Tiago S. Vaquero
Massachusetts Institute of Technology (MIT)
77 Massachusetts Avenue
Cambridge, Massachusetts 02139-4307
tvaquero@mit.edu

Mauro Vallati[*]
University of Huddersfield
Queensgate
Huddersfield, United Kingdom
HD13DH
m.vallati@hud.ac.uk

ABSTRACT

Automated planning is a prominent Artificial Intelligence challenge, as well as being a common capability requirement for intelligent autonomous agents. A critical aspect of what is called domain-independent planning, is the application knowledge that must be added to the planner to create a complete planning application. This is made explicit in (i) a domain model, which is a formal representation of the persistent domain knowledge, and (ii) an associated problem instance, containing the details of the particular problem to be solved. Both these components are used by automated planning engines for reasoning, in order to synthesize a solution plan. Formulating knowledge for use in planning engines is currently something of an ad-hoc process, where the skills of knowledge engineers significantly influence the quality of the resulting planning application. On top of that, a notion of quality of the knowledge captured within a domain model is missing; it is therefore hard to provide useful guidelines to knowledge engineers.

This paper raises some issues relating to the engineering of application knowledge for automated planning, focussing on the domain model. It uses the idea of a domain model as a formal specification of a domain, and considers what it means to measure the quality of such a specification. To do this it proposes definitions of the attributes of a domain model and its encoding language, which are needed by the automated planning community in order to improve tools for supporting the engineering of planning knowledge, and to advance toward a shared and inclusive definition of quality of domain models.

CCS CONCEPTS

• **Computing methodologies → Artificial intelligence**; **Knowledge representation and reasoning**; **Planning and scheduling**;

KEYWORDS

Automated Planning, Knowledge Engineering, Domain Model

[*]Corresponding author.

ACM Reference format:
Thomas L. McCluskey, Tiago S. Vaquero, and Mauro Vallati. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In *Proceedings of ACM Conference on Knowledge Capture, Austin, Texas USA, December 2017 (K-CAP'17),* 8 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Automated planning is a research discipline that addresses the problem of generating a totally- or partially-ordered sequence of actions that transform the environment from some initial state to a desired goal state. Automated planning has been successfully applied for decades in several areas, including space exploration [1], machine tool calibration [27], and urban traffic control [25] to mention a few.

Undoubtedly, the intensive development of domain-independent planners has contributed to the advancement of planning technology, as planning engines can be exploited as embedded components within a larger framework. Since they accept the domain and problem instance in a well defined interface language and return plans using the same syntax, they can be interchanged without any changes to the rest of the system. On the other hand, the efficiency of plan generation remains one of the most prominent challenges in artificial intelligence. Domain-independent planning engines have to deal with the complexity issues inherent in plan generation, which are exacerbated by the separation of planner logic from domain knowledge.

Knowledge Engineering in Planning and Scheduling (KEPS) was defined in the 2003 PLANET Roadmap [23], specifically for domain-independent planners, as the collection of processes involving (i) the acquisition, validation and verification, and maintenance of planning domain models, (ii) the selection and optimisation of appropriate planning machinery, and (iii) the integration of (i) and (ii) to form automated planning and scheduling (P&S) applications. KEPS can be seen as a special case of knowledge engineering, where the need for methodologies for acquiring, domain modelling and managing formally captured knowledge has long been accepted. It is also related to the area of capturing conceptual knowledge and developing domain models for Qualitative Reasoning in the general Modelling and Simulation area [4]. However, the peculiarities of automated planning and scheduling applications distinguish KEPS from general knowledge-based and simulation systems. Firstly, KEPS is concerned with the acquisition and representation of knowledge about actions, resources, processes, events, and the effect these have on a state. Secondly, this knowledge is to be used to create a system that synthesises plans, rather than performing the more common functions of knowledge systems such as classification, diagnosis or decision making. Thirdly, the knowledge is often acquired in two parts: a specification of persistent knowledge (in the literature this part is called the "domain" or "domain model") and a specification of particular scenarios representing the planning problem instance. As an illustrative example, planning has recently been applied to the area of urban traffic control[38]. In the resulting industrial-led project using real scenarios and data [25], the domain model used contains representations of continuous processes, events and actions, and the scenarios (problem instances) contain hundreds of assertions concerning road links, junctions, traffic flows and traffic signal configurations.

Studies on KEPS have led to the creation of several tools and techniques to support the design of domain knowledge structures, and the use of planners for real-world problems. Most of these tools have been presented in specialised workshops such as the *Knowledge Engineering for Planning and Scheduling*[1] workshop and the *Verification and Validation of Planning Systems*[2] workshop, as well as competitions such as the *International Competition on Knowledge Engineering for Planning & Scheduling* (ICKEPS).[3] The competitions have motivated the development of powerful KEPS systems and advances in domain modelling techniques, languages and analysis approaches. However, the 2016 edition of ICKEPS [5] also highlighted two main issues. Firstly, most teams did not use any KEPS tools (except text editors), and thus relied only on their expertise. Even more worrying is the fact that some teams were not aware of the *existence* of KEPS tools. Secondly, the number of participants of ICKEPS is still not very large, especially when compared with the latest edition of the International Planning Competition [36]: this suggests that the planning community underestimates the importance of knowledge engineering, despite its enormous impact on applicability of domain-independent planning in real-world scenarios.

Methods, algorithms, tools and representation languages to support and organise the design of such domain models are important (e.g. as shown by CommonKADS [33] in the related area of Knowledge-Based Systems), but they need to be related to commonly agreed domain model properties and metrics for the objects being designed. Within the process of domain model creation and design, this paper focuses on attempting to define a set of fundamental domain model properties –a research topic that does not seem to have progressed since much earlier work [21]. The introduction and discussion of properties is of pivotal importance for the planning and the artificial intelligence community: it allows to improve KEPS tools, for supporting the knowledge engineering of planning domain models, and to advance towards a shared and inclusive definition of quality of domain models.

The remainder of this paper is organised as follows. Section 2 contextualises and motivates the work presented in this paper. Then, in Section 3, the relevant background information on automated planning is introduced. Section 4 the concept of the domain model is defined, and in Section 5 properties and metrics are introduced and described. The implications of the introduced properties are further discussed in Section 6, and in Section 7 conclusions are given.

[1] http://icaps14.icaps-conference.org/workshops_tutorials/keps.html
[2]http://icaps11.icaps-conference.org/workshops/vvps.html
[3]http://icaps-conference.org/index.php/Main/Competitions

2 MOTIVATION

The accelerated development of AI Planning has facilitated take up of planning tools as components in autonomous systems. This take up has meant that engineering issues surrounding automated planning is of growing importance. Stakeholders, and users in particular, of autonomous systems need to be assured of the dependability of such systems. There is a need to demonstrate traditional engineering qualities within a product, such as the validity and reliability of the autonomous system. This is of paramount importance in many applications which are safety critical or safety related, in particular, in the areas of robotics and autonomous vehicles. Thus, being able to define and measure qualities of the planning-related knowledge is necessary in the process of engineering the final product.

A particular issue in the emerging aspects of agent capabilities, is their ability to be able to autonomously capture, engineer and refine knowledge. In this case, engineers who create these agents require methods, metrics and evaluation criteria to discuss the quality of the knowledge captured by the agent. A major part of this captured knowledge, to enable rational planning and reasoning, would be in the form of a domain model. In order to compare different learning algorithms one needs to be able to discuss and ultimately measure the quality of the resulting model.

Another motivation is the increasing complexity of the domain model, requiring the need to capture such knowledge in a structured and well-founded way. In multi-agent planning, for example, plans may be both generated and executed in a distributed fashion by a collection of agents. As well as the conventional dynamical aspects of knowledge about actions, the public/privacy features residing with individual agents need to be captured. Or again in planning with hybrid systems, the specification of continuous variables, continuous processes, and exogenous events needs to be captured faithfully as a necessary precondition for planning.

3 AUTOMATED PLANNING

Automated planning, and specifically classical planning, deals with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from some initial state to a desired goal state [13]. In the classical representation *atoms* are predicates. *States* are defined as sets of ground predicates. A *planning operator* $o = (name(o), pre(o), eff^-(o), eff^+(o))$ is specified such that $name(o) = op\_name(x_1, \ldots, x_k)$ (op_name is a unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator), $pre(o)$ is a set of predicates representing the operator's preconditions, $eff^-(o)$ and $eff^+(o)$ are sets of predicates representing the operator's negative and positive effects. *Actions* are ground instances of planning operators. An action $a = (pre(a), eff^-(a), eff^+(a))$ is *applicable* in a state s if and only if $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus eff^-(a)) \cup eff^+(a)$.

As stated, this knowledge is made explicit in two components: a domain model and a problem instance. When using the dominant family of planning knowledge representation languages - the PDDL[12], the planning domain model and problem instance are provided to planners as two different files, and the same domain model is used for all the problems of the application. In the restricted world view of classical planning, a domain model is specified via sets of predicates and planning operators. A *problem instance* is specified via an initial state and set of goal atoms, that need to be reached. A *solution plan* is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that satisfies the goal.

The classical planning model can be extended, in order to handle a wider range of constraints and increase expressiveness. For instance, this is the case in Temporal Planning, where actions have a duration that should be considered, or Uncertainty Planning, that studies cases in which the environment is not fully observable and effects are non-deterministic. A further extension is named PDDL+ [10], which contains constructs to define hybrid domains, including processes and events. On this matter, the interested reader is referred to [13] and [11].

4 THE DOMAIN MODEL IN AUTOMATED PLANNING

Domain Modelling is a phrase used perhaps with a variety of meanings in computer science and applied mathematics. A domain model is often described as an abstract conceptual description of some application, and is used as an aid to the software development process. It is formed as part of the requirements analysis in order to specify objects, actors, roles, attributes, etc, independent of a software implementation. A domain model is often represented imprecisely using diagrams, such as in the Unified Modelling Language (UML) [31], for "human consumption" –that is, for the benefit of analysts and developers to explore requirements and to subsequently create software in the application area being modelled.

The meaning of a domain model for representing knowledge within a planning application is much more specific. It is still an abstract conceptual description of some application area but it is encoded for a different purpose: *for the analysis, reasoning and manipulation by a planning engine in order to solve planning problems.*

Let us assume that a requirements specification for the planning component of some wider project is available. The requirements may be in the minds of domain experts, be described informally in diagrams and textual documents, or described (at least in part) in a formal language (e.g., as in the use of LTL [30]). The requirement specification would naturally contain descriptions of the kind of planning problems that the planner needs to solve, and the kind of plans that need to be provided as output. For example, it might be essential that resource consumption is taken into consideration and so plans need to be generated which achieve goals while minimising resource consumption. Before a domain-independent planner can be chosen and used, the domain information needs to be conceptualised and formalised. During this process the assumptions and features that are essential to represent a domain model are derived from the overall requirements. Within this context we define a *planning domain model* as follows:

Definition 4.1. A planning **domain model** is a formal specification of the application domain part of the requirements specification which represents entities invariant over every problem instance, such as object classes, functions, properties, relations, and operators.

```
(:action LOAD-TRUCK
  :parameters
   (?obj - obj
    ?truck - truck
    ?loc - location)
  :precondition
   (and (at ?truck ?loc) (at ?obj ?loc))
  :effect
   (and (not (at ?obj ?loc)) (in ?obj ?truck))
   )
```

Figure 1: An example of a possible encoding of a LOAD-TRUCK operator in PDDL.

This is in line with terminology from general Knowledge Engineering, specific work on domain "theories" of physical systems [4], and what is called the "domain file" in the most common planning domain encoding language, PDDL [26]. In particular, we expect the language in which the domain model is written to have a well-defined syntax and operational semantics: this means that, independent of planner and domain, there is a defined process for executing plans which correspond to sequences of actions in the application domain. In other words, there is a sound interpretation of the dynamics for any well-formed domain model [21], and such interpretations are embedded into validator tools such as VAL [17]. Adding in a problem instance, gives a knowledge model:

Definition 4.2. A planning **knowledge model** is a domain model and a corresponding problem instance

An example of PDDL encoding of part of a domain model is the *LOAD-TRUCK* operator, shown in Figure 1. Such an operator describes how the state of the world is changed (effect section) when a truck is loaded at a location *?loc*, and lists the preconditions that must be met in order to correctly apply the operator.

5 MODEL QUALITY

The central part of a domain model in AI Planning is the representation of the set of actions that a planner can reason about and the elements that support the specification of actions. It forms a potentially complex knowledge base, and its correctness is an essential factor in the overall quality of the planning function. Indeed, Bensalem et al. [3] state that domain models present the biggest validation and verification challenge to the Planning and Scheduling community. Questions include: *how should domain models be formulated, debugged and generally judged to be fit for a purpose? how should we determine the quality of the domain model and the quality of the language in which it is written? what are the criteria for choosing one domain model encoding rather than another, within that language?*

For example, it has been shown that the choice of representation within the same language of a domain –as well as the order in which elements are listed within the domain– have a significant bearing on the efficiency of planning [32, 37], and the process of domain model production may have a general bearing on quality [34]. As well as informing the general AI community, and the planning applications community, answers to these questions will form the underpinning of any tool sets to assist in this development process.

Domain models are often formulated into a planner input language directly from a requirements specification using only basic editors (these are so-called *hand-crafted* domain models), such as PDDL-studio [29] or the online editor PDDL editor [4]. In some cases, the requirements are encoded firstly into a more application-oriented language such as UML in itSIMPLE [40], or in AIS-DDL in the KEWI interface [41], and then mapped into the target planner's input language. The international competition on knowledge engineering for automated planning and scheduling in 2009[5] for example, was dedicated to evaluating systems that expect the requirements to be captured in an application-oriented notation, then the domain model is produced automatically or semi-automatically. Hence, in this case, the quality of the domain model is dependent both on the initial encoding and the correctness of the translation process.

As a variation on this, domain models can be formulated by automated acquisition tools, such as in the LAMP system [42], and the LOCM system [6]. These type of systems acquire domain models from example plans with little or no pre-engineered domain knowledge. The LAMP system can form PDDL domain models from example plan scripts and associated initial and goal states. It is aimed at helping knowledge engineers create a new domain model, as the authors maintain that, after learning, the model needs to be hand-crafted to remove bugs. Another system, LOCM, exploits the assumption of an object-centred domain to enable it to learn from plan scripts only. As with LAMP, LOCM outputs a model in a PDDL format, but it inputs only training plan scripts: it does not require representations of initial and goal states, or any descriptions of predicates, object classes, states etc. In the context of an engineering process, training plans would therefore be considered part of the requirements specification: they would be examples of the kinds of plans that the system would be required to generate. For this class of learning system, the concept of quality of the domain model is key to being able to evaluate the learning process.

Assessing the quality of the domain model is naturally part of the verification and validation (V&V) processes of the overall planning system. The importance of V&V in domain models for planning has long been recognised in both domain independent work [24] and more specifically in space applications [28]. Ways to assess the quality of a domain model can be classed into two types – dynamic and static– in a similar way to investigating program code quality. In practice, debugging and validating the domain model is invariably done using dynamic testing, i.e testing the ability to execute a planner with a domain model and a particular problem instance. This dynamic view of V&V of the domain model is taken in the work of Bensalem et al. [3]. The authors focus on model checking, a method often used to test formal specifications, that exhaustively checks all reachable states, to test whether a path can be found to a goal. Model checking is not feasible for larger

[4]http://editor.planning.domains/
[5]ICKEPS 2009. http://kti.mff.cuni.cz/ bartak/ICKEPS2009/

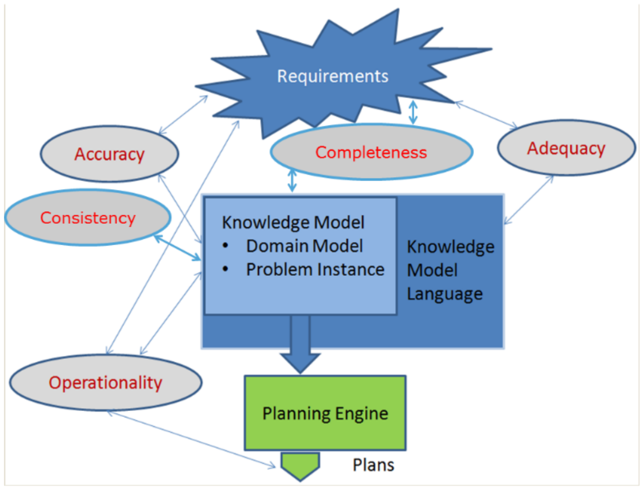

Figure 2: An overview of the introduced domain model properties, and how they relate with domain requirements, planning engine, (potentially) generated plans, the current knowledge model, and the exploited language.

domain models, however, as the state space to be explored can be astronomical. What states may be reachable depends on the starting position of the domain model checking process. In other words, the process works with a fixed initial state, whereas in AI Planning we need to investigate the domain model over a range of initial states.

Here we consider *static* as well as *dynamic* properties of syntactically correct domain models, and through these, introduce notions of quality. In the style of software quality domain models and software metrics, we define three attributes of domain models (accuracy, consistency, and completeness), one of a domain model's encoding language (adequacy), and one of a domain model-planner pairing (operationality). These attributes (and metrics related to them) are intended to be used to investigate issues in the engineering of domain models, to be embedded in knowledge engineering tools, and hence to contribute to the overall validation and verification of the planning system.

Figure 2 shows the introduced domain model properties, and the way in which they relate with the aspects and elements involved in the knowledge engineering process of planning applications, i.e., the domain requirements, the planning engine, the resulting (potentially) generated solutions plans, the current knowledge model, and the exploited model language.

5.1 Consistency and Accuracy

Let us assume the knowledge model contains a set of logical expressions. We will take a model-theoretic approach to the semantics of domain models and problem instances, and use this to form the property definitions as shown in Figure 2. An interpretation of the planning knowledge model is a mapping of its components to features (objects, relations, etc.) in some application domain. Facts in an initial state then become simple assertions, and ground operators then become more complex assertions about when it is possible to execute an action, and what effects the action has, in the application domain. Clearly we are interested in knowledge models that have an interpretation given by the requirements specification that makes the assertions in the model true.

Definition 5.1. A knowledge model is **consistent** if there exists *at least one* interpretation that makes all its assertions true.

Consistency is checking whether there is *any* interpretation that can make the planning domain knowledge true, and therefore can be considered simply as a property of the knowledge model. Consistency checks are therefore application-independent. They can spot obvious but common errors, e.g. if the domain knowledge admits a state which contains both a fact and the fact's negation, then no interpretation can make the state true, and hence no interpretation can satisfy it. Or again, if the operator in Figure 1 contained as an additional precondition the term *(not (at ?obj ?loc))* then no interpretation could satisfy it, and hence the operator could never be grounded into a usuable action.

We can widen this definition to **accuracy** by considering the truth of the assertions when the interpretation is given by the requirements specification of the application at hand.

Definition 5.2. A knowledge model **accurately represents** part of the requirement specification if the interpretation given to it by mapping its components to features (objects, relations, etc.) in that specification, makes all assertions in the knowledge model true.

Consistency is thus a special case of accuracy. Accuracy is an attribute of the knowledge model, related to application domain features considered as part of the requirements specification.

Verifying accuracy is essentially an informal process if the requirements are described informally. If the requirements are already encoded in some formal language, then a knowledge model is accurate if the requirements provide a *model* of the knowledge (in the sense that the knowledge model is an abstract algebra, and the requirements a concrete algebra).

5.2 Use of the Properties in PDDL encodings

In PDDL, a knowledge model is encoded as a domain file and a problem file. The dynamics that will allow the planner to generate a solution plan, that when executed from the initial state can reach the goal state, are described in the domain model file. To check the accuracy of operators in this file we could:

- create all possible reachable groundings of operators, using the objects in the problem file;
- map the logical expression in the precondition of each grounded schema to a set P of relations and properties in the requirements;
- map the logical expression in the effects to a set E of relations and properties in the application; and
- check that if P is true in the application, then the action domain modelled could be executed, and if executed would make E true.

A similar process would be used to assess the accuracy of the problem instance contained in the problem file: it is a matter of checking that the initial state and goal maps to the problem embedded in the requirements specification.

If a domain model is inaccurate, then it may be possible to create metrics to measure the inaccuracy. For example, in a PDDL domain model learning system such as LAMP [42], there is a need to evaluate the quality of the domain model learned, as part of the learning tools' evaluation. Using a comparison with a hand-crafted domain model, the authors judge the accuracy of the domain model by counting the number of missing predicates in the learned domain model. To relate it to our definition, the hand-crafted domain model takes the role of part of the requirements –in this case these requirements are themselves stated precisely in a formal language– though it is a matter for future research to discover sound ways of measuring differences when comparing two domain models [35].

As a rule of thumb, if all the considered problem instances are shown to be inconsistent in a similar way, it may be argued that the domain model is inconsistent as well. On the other hand, if the consistency of a domain model is merely assessed by analysing the consistency of problems, the fact that no inconsistencies have been identified in such a set of considered problems, is not sufficient for claiming that the domain model is consistent for every possible problem from the domain.

5.3 Completeness

In software engineering, the completeness of formal specifications has been recognised as a difficult but important topic that is best tackled in specific types of application (e.g., Leveson deals with process control systems using a check-list [19]). For automated planning, we introduce the following definition:

Definition 5.3. Given a specific problem instance and a domain model, and I their interpretation mapping to the requirements specification, then the domain model is **complete** if the following conditions hold:

- (i) for any solution plan S for problem instance P that can be formed from ground operators in the domain model, $I(S)$ is an acceptable solution for $I(P)$ in the requirements;
- (ii) the converse is also true, that is, for any acceptable solution S' to problem $I(P)$ there exists a solution plan S derivable from the domain model for problem P such and $I(S) = S'$.

In practice, the requirements would contain a set of problems $P_1..P_n$ requiring solution for a fixed domain model (DM), hence each $P_i DM$ would need to be complete in this sense. As an example, failing to include operator schema in a domain model which are necessary to provide solutions, cause incompleteness. Using these definitions, it is possible to find domain models which are complete but not accurate. This is so because, given a complete domain model, it would be possible to add a construction (e.g., an extra operator) that interpreted to something false in the requirement specification, but which never interfered in the solutions of required problems to be solved.

5.4 Adequacy

Adequacy is a relationship between the requirements and the language in which the domain model is encoded. Given the fact that requirements cannot be changed or (usually) easily modified, we define the adequacy on the basis of the chosen language.

Definition 5.4. A language is **adequate** with respect to a requirements specification if it has the expressive power to represent the requirements within a domain model in sufficient detail so that a complete domain model can be expressed.

Adequacy is related to the level of granularity needed by the requirements, and derives from the idea of representational or expressive adequacy of a knowledge representation language. As many extensions of PDDL have been introduced in the last decades, each of them extending the expressive power of the language, an obvious example of inadequacy is where we might use basic PDDL, that does not allow to represent any sort of numerical or temporal information, but where the requirements demand that actions have a specific temporal duration.

It should be noted that the completeness of a domain model depends on language adequacy. A domain model could be accurate (all the features present conform to the requirements, in the sense that their interpretation is true), but it may be the case that some requirements cannot be represented at all. Hence, the completeness of a domain model may be prevented because of an inadequate language.

5.5 Operationality

In the automated planning and scheduling literature, the validation of a domain model is often solely based on a test of whether it will lead to acceptable behaviour in a planning and scheduling or application system [20, 34], that is, if an acceptable plan can be output. In fact, given the definition of completeness we introduced in Section 5.3, the above mentioned informal definition of operationality can be seen as a weak form of completeness. However, there are normally many encodings of any given domain model that would pass this test, but some encodings lead to much more efficiently generated solutions than others. In our applied work on Urban Traffic Control, for example, we showed that, using the *same* planner, improving the efficiency of the encoding considerably enhanced the scope of the problem instances that the planner could solve [25].

Given that a complete domain model exists, there will always be ways of re-representing the domain model without compromising completeness. These domain models may give different results when input to a planner: for example, some may not satisfy some real time constraints in the requirements of the application domain. In this sense, operationality is also a relationship partly dependent on the choice of the planning engine.

More generally, it is also possible that two distinct domain models are complete, but one leads to a more efficient implementation, or better quality plans. Hence, the process of finding an acceptable plan in an application depends not only on the strategy used by the planner, but also the domain model. In particular, if the model is not accurate, then the planner will generate flawed plans or no plans at all.

The speed of plan generation can be affected dramatically under such circumstances. For instance, case studies have shown that fixing and refining the domain model itself (e.g., adding additional relevant knowledge) can improve the performance of planners, without modifying the planners and their search mechanism [39]. In addition, works like [2, 7, 8, 18] show that adding relevant, redundant constraints (in the form of control knowledge and rules) in the domain model can also speed up planning engines, thus provide a solution plan in a shorter CPU-time. Moreover, it is worth reminding that CPU-time is not the only resource that can be constrained in requirements: RAM and hard drive usage, for instance, can represent critical resources as well.

For a given planner and requirements specification, we define *Operationality* as an attribute of a domain model and a planning engine E as follows.

Definition 5.5. A domain model is **operational** with respect to planning engine E if E produces a solution S to a problem instance P with acceptable resource bounds, such that $I(S)$ is an acceptable solution to $I(P)$ according to the requirements.

In this context, acceptable resource bounds can be defined –for instance– in terms of runtime, memory usage, number of CPUs, etc. Of course, it is assumed that such resource bounds allow the planning engine E to run properly. From this perspective, the operationality property of a domain model can also change over time, as acceptable resource bounds may be reduced –due to modified application requirements–, the planning engine can be substituted, or the machine on which the planning engine is executed may be changed. Since planning engines' performance can be significantly affected by the hardware and software configuration of the exploited machine [16], it may also be the case that exogenous events –such as a major release of a compiler, or a new version of an operating system– can impact on the operationality of a given domain model, with regards to a considered planning engine. Note that the definition does not demand that the planner outputs *all* the acceptable solutions; this is somewhat infeasible computationally, hence we have a weaker definition that is more in tune with practice.

Many state-of-the-art domain-independent PDDL planners automatically investigate operationality (as well as performing checks that help promote confidence in the consistency or accuracy of a knowledge model). They exploit a pre-processing step –such as the reachability analysis– for assessing whether the specified goals can be achieved by exploiting the available actions, and objects described in the initial state. While this pre-processing step can be incomplete, for the sake of planning efficiency, recent works have been focused on providing formal and complete unsolvability proofs, under the form of *certificates*, of a considered planning problem [9]. Furthermore, Göbelbecker et al. [14] introduced an approach that, given a problem that is unsolvable (i.e., inconsistent), can suggest how to modify the model in order to make it solvable. Finally, there are tools such as Torchlight [15] which focus on investigating the shape of the search space for some well-known heuristics approaches. This sort of techniques can already be fruitfully exploited for obtaining an overview of the consistency of a problem model, described in PDDL and, implicitly, to investigate also the domain model.

6 DISCUSSION

One key question is how well a modelling language supports the pursuit of forming high quality knowledge models, and particularly domain models. Do the language features support the faithful engineering of a model? PDDL, for example, was shown to not be too good in this case [22] primarily because there are few inbuilt

```
(:process keepinter
  :parameters (?p - phase ?i - intersection)
  :precondition (and
      (inter ?p) (contains ?i ?p)
      (< (intertime ?i) (interlimit ?p)  )
   )
   :effect (and
      (increase (intertime ?i) (* #t 1 ) )
   ))
```

Figure 3: An example of a process in PDDL+ which models time passing between phases of traffic signals in a road intersection (the time between phases is called the inter-green time)

constructs such as state invariants which help cut down on bugs. *The weakness in using PDDL for knowledge engineering is that there is typically much meta knowledge left implicit that governs what is an acceptable state or problem instance in an application, and what is not, which cannot be captured by the PDDL. In an application, this meta knowledge can be stated as a set of invariants on planning states, and proves very useful in checking the validity of problem instances, but must be stated within another formalism.*

Another question is, do the definitions of the key criteria easily generalise to more expressive languages.

Consider PDDL+. this is a language for encoding domain models which contain processes, events and continuous variables, as well as actions and discrete variables. An example of a simple PDDL+ process definition is given in Figure 3. This models the passing of the inter-green time between two consecutive phases of traffic signals in a road intersection. While the general definition of accuracy for a PDDL+ model is still appropriate, the procedure to check PDDL given relies on there being a finite grounding of operators, which is not the case for a hybrid language containing continuous variables. Also, the operational semantics of PDDL+ is much more complex than that of classical planning, with problems to do with co-occurrence of actions and processes, and cascading of events, as is found in hybrid automata [10]. Thus while the properties defined in the paper are still useful, checking the quality of such expressive models is more difficult.

Consider the case of Multi Agent Planning: For multi-agent plan generation, the additional machinery required is the public/privacy nature of features (predicates, actions). There have been several schemes on how to deal with this, such as the use of features being either global to all agents, or restricted to one agent. We could localise the quality criteria to each agent, then look at the overall quality in terms of the sum of the individual. Plan generation can be conceptualised as each agent creating their own complete plan, and the public part of these plans being combined by a central agent in order to achieve the main goal. In this case, in contrast to the case for PDDL+, there needs to additional quality criteria to deal with the communication and/or privacy aspects. Additionally, concepts such operationally are more complex, since the execution of plans depends on more than one agent.

7 CONCLUSION

In this paper we have investigated the role and nature of the planning domain model, and discussed its importance in the process of creating a planning application.

Notions of the "quality" of a domain model are needed for many reasons, not least (i) to help engineers construct models, (ii) to underpin tools and environments that help in the process of creating models, (iii) to assist in the efficiency of planning, (iv) to assess action learning programs, (v) and to compare one domain model against another.

We have introduced notions of consistency, accuracy, completeness, adequacy and operationality with respect to the formation of planning domain models and their associated problem instances. We have introduced a much-needed discussion on the subject, and explored the implications of these definitions as aspects of the quality of the planning knowledge model. Our future work will further explore these definitions and how they can be used to compare and contrast existing KEPS tools and methods.

REFERENCES

[1] Mitchell Ai-Chang, John Bresina, Len Charest, Adam Chase, JC-J Hsu, Ari Jonsson, Bob Kanefsky, Paul Morris, Kanna Rajan, Jeffrey Yglesias, et al. 2004. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems* 19, 1 (2004), 8–12.

[2] Fahiem Bacchus and Froduald Kabanza. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116, 1-2 (2000), 123–191.

[3] Saddek Bensalem, Klaus Havelund, and Andrea Orlandini. 2014. Verification and validation meet planning and scheduling. *International Journal on Software Tools for Technology Transfer* 16, 1 (2014), 1–12.

[4] Bert Bredeweg, Paulo Salles, Anders Bouwer, Jochem Liem, Tim Nuttle, Eugenia Cioaca, Elena Nakova, Richard Noble, Ana Luiza Rios Caldas, Yordan Uzunov, et al. 2008. Towards a structured approach to building qualitative reasoning models and simulations. *Ecological Informatics* 3, 1 (2008), 1–12.

[5] Lukáš Chrpa, Thomas L McCluskey, Mauro Vallati, and Tiago S Vaquero. 2017. The Fifth International Competition on Knowledge Engineering for Planning and Scheduling: Summary and Trends. *AI Magazine* 38, 1 (2017), 104–106.

[6] Stephen Cresswell, Thomas L. McCluskey, and Margaret M. West. 2013. Acquiring planning domain models using LOCM. *Knowledge Engineering Review* 28, 2 (2013), 195–213.

[7] Tomás de la Rosa and Sheila McIlraith. 2011. Learning domain control knowledge for TLPlan and beyond. In *ICAPS 2011 Workshop on Planning and Learning*. 36–43.

[8] Patrick Doherty and Jonas Kvarnstram. 2001. TALplanner: A temporal logic-based planner. *AI Magazine* 22, 3 (2001), 95.

[9] Salome Eriksson, Gabriele Röger, and Malte Helmert. 2017. Unsolvability Certificates for Classical Planning. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS)*.

[10] Maria Fox and Derek Long. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research* 27 (2006), 235–297.

[11] Hector Geffner and Blai Bonet. 2013. *A Concise Introduction to Models and Methods for Automated Planning.* Morgan & Claypool Publishers.

[12] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. *PDDL - the planning domain definition language.* Technical Report CVC TR-98-003/DCS TR-1165. Yale Center for Computational Vision and Control.

[13] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers.

[14] Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. 2010. Coming Up With Good Excuses: What to do When no Plan Can be Found. In *Proceedings of the International Conference on Automated Planning and Scheduling ICAPS*. 81–88.

[15] J. Hoffmann. 2011. Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and h+. *Journal of Artificial Intelligence Research* 41 (2011), 155–229.

[16] Adele E. Howe and Eric Dahlman. 2002. A Critical Assessment of Benchmark Comparison in Planning. *Journal of Artificial Intelligence Research* 17, 1 (2002), 1–33.

[17] Richard Howey, Derek Long, and Maria Fox. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings*

of the 16th IEEE International Conference on Tools with Artificial Intelligence. 294–301.

[18] Yi-Cheng Huang, Bart Selman, and Henry Kautz. 1999. Control Knowledge in Planning: Benefits and Tradeoffs. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence, Orlando, FL.*

[19] Nancy Leveson. 2000. Completeness in formal specification language design for process-control systems. In *Proceedings of the third workshop on Formal methods in software practice.* 75–87.

[20] Derek Long, Maria Fox, and Richard Howey. 2009. Planning domains and plans: validation, verification and analysis. In *Workshop on V&V of Planning and Scheduling Systems.*

[21] T. L. McCluskey. 2002. Knowledge Engineering: Issues for the AI Planning Community (Keynote Talk). In *Proceedings of the AIPS Workshop on Knowledge Engineering Tools and Techniques for AI Planning.*

[22] T. L. McCluskey. 2003. PDDL: A Language with a Purpose?. In *Proc. PDDL Workshop, ICAPS, Trento, Italy.*

[23] T. L. McCluskey, R. Aler, D. Borrajo, M.Garagnani, P.Haslum, P. Jarvis, I.Refanidis, and U. Scholz. 2003. Knowledge Engineering for Planning Roadmap. *http://scom.hud.ac.uk/planet/home* (2003).

[24] T. L. McCluskey and J. M. Porteous. 1997. Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency. *Artificial Intelligence* 95 (1997), 1–65.

[25] Thomas L. McCluskey and Mauro Vallati. 2017. Embedding Automated Planning within Urban Traffic Management Operations. In *Proceedings of the International Conference on Automated Planning and Scheduling ICAPS.*

[26] Drew Mcdermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. *PDDL - The Planning Domain Definition Language.* Technical Report. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

[27] Simon Parkinson and Andrew Peter Longstaff. 2015. Multi-objective optimisation of machine tool error mapping using automated planning. *Expert Syst. Appl.* 42, 6 (2015), 3005–3015.

[28] J. Penix, C. Pecheur, and K. Havelund. 1998. Using Model Checking to Validate AI Planner Domain Models. In *Proceedings of the 23rd Annual Software Engineering Workshop.*

[29] Tomas Plch, Miroslav Chomut, Cyril Brom, and Roman Barták. 2012. Inspect, Edit and Debug PDDL Documents: Simply and Efficiently with PDDL Studio. *ICAPS12 System Demonstration* (2012), 4.

[30] Franco Raimondi, Charles Pecheur, and Guillaume Brat. 2009. PDVer, a tool to verify PDDL planning domains. (2009).

[31] Manny Rayner, Beth Ann Hockey, Nikos Chatzichrisafis, and Kim Farrell. 2005. OMG Unified Modeling Language specification. In *Version 1.3,© 1999 Object Management Group, Inc.*

[32] Patricia J Riddle, Robert C Holte, and Michael W Barley. 2011. Does Representation Matter in the Planning Competition?. In *Proceedings of SARA.*

[33] Guus T. Schreiber and Hans Akkermans. 2000. *Knowledge Engineering and Management: The CommonKADS Methodology.* MIT Press.

[34] Mohammad Munshi Shahin Shah, Lukas Chrpa, Diane E Kitchin, Thomas Leo McCluskey, and Mauro Vallati. 2013. Exploring Knowledge Engineering Strategies in Designing and Modelling a Road Traffic Accident Management Domain.. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

[35] S Shoeeb and TL McCluskey. 2011. On comparing planning domain models. In *Proceedings of the Annual PLANSIG workshop.* 92–94.

[36] Mauro Vallati, Lukáš Chrpa, Marek Grzes, Thomas L McCluskey, Mark Roberts, and Scott Sanner. 2015. The 2014 International Planning Competition: Progress and Trends. *AI Magazine* 36, 3 (2015), 90–98.

[37] Mauro Vallati, Frank Hutter, Lukáš Chrpa, and Thomas L McCluskey. 2015. On the Effective Configuration of Planning Domain Models. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

[38] Mauro Vallati, Daniele Magazzeni, Bart De Schutter, Lukáš Chrpa, and Thomas L McCluskey. 2016. Efficient Macroscopic Urban Traffic Models for Reducing Congestion: a PDDL+ Planning Approach. In *The Thirtieth AAAI Conference on Artificial Intelligence (AAAI).* 3188–3194.

[39] Tiago Stegun Vaquero, José Reinaldo Silva, J Christopher Beck, et al. 2010. Improving planning performance through post-design analysis. In *Proceedings of the 2010 workshop on Scheduling and Knowledge Engineering for Planning and Scheduling (KEPS).* 45–52.

[40] Tiago S Vaquero, José R Silva, Flavio Tonidandel, and J Christopher Beck. 2013. itSIMPLE: towards an integrated design system for real planning applications. *The Knowledge Engineering Review* 28, 02 (2013), 215–230.

[41] Gerhard Wickler, Lukás Chrpa, and Thomas Leo McCluskey. 2014. KEWI-A Knowledge Engineering Tool for Modelling AI Planning Tasks.. In *Proceedings of KEOD.* 36–47.

[42] H.H. Zhuo, Q. Yang, D. H. Hu, and L. Li. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174, 18 (2010), 1540 – 1569. https://doi.org/DOI:10.1016/j.artint.2010.09.007