Issues in Planning Domain Model Engineering

Thomas L. McCluskey
School of Computing and Engineering
University of Huddersfield
t.l.mccluskey@hud.ac.uk

Tiago S. Vaquero
California Institute of Technology
Massachusetts Institute of Technology
tstegunv@caltech.edu

Mauro Vallati
School of Computing and Engineering
University of Huddersfield
m.vallati@hud.ac.uk

Abstract

The paper raises some issues relating to the engineering of domain models for automated planning. It studies the idea of a domain model as a formal specification of a domain, and considers properties of that specification. It proposes some definitions, which the planning and, more generally, the artificial intelligence community needs to consider, in order to properly deal with engineering issues in domain model creation.

1 Introduction

Knowledge Engineering in Planning and Scheduling (KEPS) was defined in the 2003 PLANET Roadmap (McCluskey et al. 2003), 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 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 (Bredeweg et al. 2008a). However, the peculiarities of AI P&S applications distinguish KEPS from general knowledge-based and simulation systems. Firstly, KEPS is concerned with the acquisition and representation of knowledge about actions, 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 making a diagnosis or decision.

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 & 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.

This workshop paper focuses on KEPS research where it concerns the creation of domain models. Methods, algorithms, tools and representation languages to support and organise the design of such domain models are important (e.g. as shown by CommonKADS 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. While there have been regular KEPS workshops and ICKEPS competitions over the last 15 years, the specific area of domain model quality does not seem to received much attention since our earlier work (McCluskey 2002).

2 The Domain Model in AI 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) (OMG 2011), 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 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

[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

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 (Raimondi, Pecheur, and Brat 2009)). 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 (elaborated in the sections below) 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:

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

This is in line with terminology from general Knowledge Engineering, specific work on domain "theories" of physical systems (Bredeweg et al. 2008b), and is similar to what is called the "domain file" in the most common planning domain encoding language, PDDL (Ghallab et al. 1998) (though invariant information may also be included in the "problem file"). In particular, we expect the language in which the domain model is written to have a well-defined syntax and operational semantics: in other words, 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 (McCluskey 2002), and such interpretations are embedded into validator tools such as VAL (Howey, Long, and Fox 2004).

3 Model Quality

The central part of a domain model 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 (Bensalem, Havelund, and Orlandini 2014) state that domain models present the biggest validation and verification challenge to the P&S community.

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? In particular, 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 (Vallati et al. 2015; Riddle, Holte, and Barley 2011), and the process of domain model production may have a general bearing on quality (Shah et al. 2013). As well as informing 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 (Plch et al. 2012) 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 (Vaquero et al. 2011), or in AIS-DDL in the KEWI interface (Wickler, Chrpa, and McCluskey 2014), and then mapped into the target planner's input language. The international competition on knowledge engineering for P&S 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 in the LAMP system (Zhuo et al. 2010), and the LOCM system (Cresswell, McCluskey, and West 2010).

Assessing the quality of the domain model is naturally part of the verification and validation (V&V) processes of the overall planning system. Though this area has not received much attention in the research community, the importance of V&V in domain models for planning has long been recognised in both domain independent work (McCluskey and Porteous 1997) and more specifically in space applications (Penix, Pecheur, and Havelund 1998). 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. This dynamic view of V&V of the domain model is taken in the work of Bensalem et al (Bensalem, Havelund, and Orlandini 2014). 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 domain models, however, as the state space to be explored can be astronomical.

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 (opera-

[4] http://editor.planning.domains/

[5] ICKEPS 2009. http://kti.mff.cuni.cz/ bartak/ICKEPS2009/

tionality). 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.

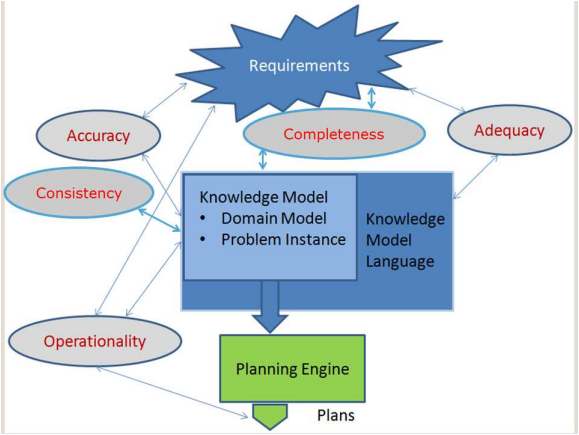

Figure 1: domain model properties

Accuracy Accuracy is an attribute of the domain model, related to application domain features considered as part of the requirements specification (often just referred as "requirements" below). Considering the domain model as a logical expression,

> *a domain model accurately represents 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 domain model true.*

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

Using the definition of domain model above, checking that the model is accurate entails checking the veracity of assertions in every state. If a domain model is encoded as a domain file in PDDL, then assessing the accuracy can be done in part by utilizing a problem file. We then need to: (i) create all possible groundings of operator schema, using the objects in the problem file; (ii) map the logical expression in the precondition of each grounded schema to a set P of relations and properties in the requirements; (iii) map the logical expression in the effects to a set E of relations and properties in the application; and (iv) 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 file. Assessing accuracy of the problem specification is a matter of checking that the initial state and goal maps to the problem embedded in the requirements specification.

Although we have defined accuracy here as a binary property, if a domain model is inaccurate, then it may be possible to create metrics to measure the inaccuracy. For example, in a domain model learning system such as LAMP (Zhuo et al. 2010), 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 (Shoeeb and McCluskey 2011).

Consistency Consistency of domain models is considered here as a special case of accuracy:

> *A domain model is consistent if some interpretation exists that makes all its assertions true.*

While accuracy is concerned with the interpretation given by the requirements being true, consistency is checking whether there is *any* interpretation that can make it true. Consistency is independent of requirements, and therefore can be considered simply as a property of the domain model itself. Hence, efficient consistency checks can spot obvious but common errors, e.g. if a domain model contains an operator that adds a fact and the negation of that same fact, then no interpretation can make it true[6]. Another example is inspired by formal software specification methods: having defined operators, and invariants for the domain, then to check for consistency we can show that under any application of an operator, the invariant is preserved (otherwise it would be possible to generate an inconsistent state, that is one that makes an invariant false). This kind of consistency check was implemented in the GIPO knowledge engineering environment (Simpson et al. 2001).

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 checklist (Leveson 2000)). For automated planning, we make the following definition:

> *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: (i) for any solution plan S for problem P that can be formed from ground operators in the domain model, $I(S)$ is an acceptable solution for $I(P)$ in the requirements; and (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*

[6] note that this is logical consistency in that no state can be consistent where a proposition and it negation is asserted. In operation, some planners may delete propositions and then add propositions, leaving the implemented state consistent.

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_iDM 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.

Adequacy Adequacy is a relationship between the requirements and the language in which the domain model is encoded.

> *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.

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. An obvious example of inadequacy is where we might use basic PDDL, but where the requirements demand that actions are durative.

Operationality In the AI Planning & 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 P&S system (Shah et al. 2013), that is, if an acceptable plan can be output. This is a weak form of completeness as defined above. 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. Given 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. 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. For example, if the model is not accurate, then the planner will generate flawed plans or no plans at all. Even the planner's speed can be affected 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 (Vaquero, Silva, and Beck 2010). In addition, works like (Huang, Selman, and Kautz 1999; Bacchus and Kabanza 2000; de la Rosa and McIlraith 2011; Doherty and Kvarnström 2001) show that adding relevant, redundant constraints (in the form of control knowledge and rules) in the domain model can also speed up planners.

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

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

Note that the definition does not demand that the planner outputs *all* the acceptable solutions; this is somewhat unfeasible computationally, hence we have a weaker definition that is more in tune with practice.

4 Discussion

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

- to help engineers construct models,
- to underpin tools and environments that help in the process of creating models,
- to assist in the efficiency of planning,
- to assess action learning programs,
- to compare one domain model against another.

To introduce a much-needed community discussion on the subject, we have developed some definitions of qualities of domain models. Our future work will attempt to fully evaluate these definitions, and explore how they can be used to compare and contrast existing KEPS tools and methods.

References

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

Bensalem, S.; Havelund, K.; and Orlandini, A. 2014. Verification and validation meet planning and scheduling. *International Journal on Software Tools for Technology Transfer* 16(1):1–12.

Bredeweg, B.; Salles, P.; Bouwer, A.; Liem, J.; Nuttle, T.; Cioaca, E.; Nakova, E.; Noble, R.; Caldas, A. L. R.; Uzunov, Y.; Varadinova, E.; and Zitek, A. 2008a. Towards a structured approach to building qualitative reasoning models and simulations. *Ecological Informatics* 3(1):1–12.

Bredeweg, B.; Salles, P.; Bouwer, A.; Liem, J.; Nuttle, T.; Cioaca, E.; Nakova, E.; Noble, R.; Caldas, A. L. R.; Uzunov, Y.; Varadinova, E.; and Zitek, A. 2008b. Towards a structured approach to building qualitative reasoning models and simulations. *Ecological Informatics* 3(1):1–12.

Cresswell, S.; McCluskey, T.; and West, M. M. 2010. Acquiring planning domain models using LOCM. In *Knowledge Engineering Review*, 1–18. Cambridge University Press.

de la Rosa, T., and McIlraith, S. A. 2011. Learning Domain Control Knowledge for TLPlan and Beyond. In *Proceedings of the ICAPS-11 Workshop on Planning and Learning (PAL)*.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A Temporal Logic-Based Planner. *AI Magazine* 22(3):95–102.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The Planning Domain Definition Language.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, 294–301. Washington, DC, USA: IEEE Computer Society.

Huang, Y.-C.; Selman, B.; and Kautz, H. A. 1999. Control Knowledge in Planning: Benefits and Tradeoffs. In Hendler, J., and Subramanian, D., eds., *AAAI/IAAI*, 511–517. AAAI Press / The MIT Press.

Leveson, N. 2000. Completeness in formal specification language design for process-control systems. In *Proceedings of the Third Workshop on Formal Methods in Software Practice*, FMSP '00, 75–87. New York, NY, USA: ACM.

McCluskey, T., and Porteous, J. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence* 95(1):1–65. © Elsevier.

McCluskey, T.; Aler, R.; Borrajo, D.; Haslum, P.; Jarvis, P.; Refanidis, I.; and Scholz, U. 2003. Knowledge Engineering for Planning Roadmap. Available at http://scom.hud.ac.uk/planet/home/.

McCluskey, T. L. 2002. Knowledge Engineering: Issues for the AI Planning Community. In *Proceedings of the AIPS-2002 Workshop on Knowledge Engineering Tools and Techniques for AI Planning, Toulouse, France*.

OMG. 2011. *OMG Unified Modeling Language Specification, Version 2.4*.

Penix, J.; Pecheur, C.; and Havelund, K. 1998. Using model checking to validate ai planner domain models. *the Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard*.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug pddl documents: Simply and efficiently with pddl studio. *ICAPS12 System Demonstration* 4.

Raimondi, F.; Pecheur, C.; and Brat, G. 2009. Pdver, a tool to verify pddl planning domains.

Riddle, P. J.; Holte, R. C.; and Barley, M. W. 2011. Does representation matter in the planning competition? In Genesereth, M. R., and Revesz, P. Z., eds., *SARA*. AAAI.

Shah, M. M. S.; Chrpa, L.; Kitchin, D. E.; McCluskey, T. L.; and Vallati, M. 2013. Exploring knowledge engineering strategies in designing and modelling a road traffic accident management domain. In Rossi, F., ed., *IJCAI*. IJCAI/AAAI.

Shoeeb, S., and McCluskey, T. 2011. On comparing planning domain models. In *The 29th Workshop of the UK Planning and Scheduling Special Interest Group PlanSIG 2011*, 92–94. UK PLANNING AND SCHEDULING Special Interest Group.

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

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. L. 2015. On the effective configuration of planning domain models. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI press.

Vaquero, T. S.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2011. itSIMPLE: Towards an Integrated Design System for Real Planning Applications. In *To appear in: The Knowledge Engineering Review Journal, special issue on International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS): Cambridge University Press*.

Vaquero, T.; Silva, J.; and Beck, J. 2010. Improving Planning Performance Through Post-Design Analysis. In *Proceedings of the ICAPS'10 Workshop on Knowledge Engineering for Planning and Scheduling. Toronto, Canada*, 45–52.

Wickler, G.; Chrpa, L.; and McCluskey, T. 2014. Kewi: A knowledge engineering tool for modelling ai planning tasks. In *6th International Conference on Knowledge Engineering and Ontology Development*.

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