Towards Explanation-Supportive Knowledge Engineering for Planning

Mauro Vallati and **Thomas L. McCluskey**
University of Huddersfield
Huddersfield, UK

Lukáš Chrpa
Czech Technical University &
Charles University in Prague, Prague, CZ

Abstract

In real-world planning applications, it is of pivotal importance that decisions and solutions can be explained, in order to maximise safety and increase the reliability of the planning component. There is a significant strand of research that focuses on exploiting planning models, and the planning process itself, in order to deal with a wide range of possible requests of explanation. However, there are explanations that can not be provided by considering only those elements, but that require additional knowledge –which is usually available during the knowledge engineering process of designing planning models.

In this paper, we introduce two classes of explanations that require a formal (or semi-formal) encoding of additional knowledge. We then describe how knowledge engineering tools can be extended in order to support the collection of such additional knowledge, and we briefly introduce the structure of an appropriate architecture.

Introduction

Automated planning has been successfully applied for decades, and is gaining importance, in several areas, including space exploration (Ai-Chang et al. 2004), machine tool calibration (Parkinson and Longstaff 2015), and urban traffic control (McCluskey and Vallati 2017) to mention a few. In applications, particularly in safety-critical ones, it is of paramount importance that generated solutions can be thoroughly explained and described. This is usually termed as *explainability*, and allows to increase the reliability of the planning component, and can support the validation of models and of automated reasoning.

There has been a growing strand of research focusing on explainability in automated planning. Works in the area cover the explanation and description of generated plans (Sohrabi, Baier, and McIlraith 2011; Caminada et al. 2014). A more recent line of work is focused on explaining also the planning process, by motivating and describing decisions taken by the planner during the exploration of the search space (Fox, Long, and Magazzeni 2017). This sort of analysis allows to provide answers to questions like *Why is this action performed? Why this other action has been ignored?*

Mentioned approaches are mainly focused on explaining plans and planning processes, and do not "question" the provided domain model that is used as the basis of the reasoning process. Therefore, any provided explanation is bounded to the model. In other words, a user would not be able to ask –or to get satisfying answers to– questions that involves a critical analysis of the model, but only questions about the search space that the considered model generates. On this matter, the work of Chakraborti et al. (2017) proposes to exploit explanations for dealing with the model reconciliation problem. Their approach takes explicitly into account models, and exploits explanation in order to minimise the divergence between the human mental model and the actual PDDL domain model. This can be done by considering only optimal plans. In a nutshell, if optimally generated plans are not self-explanatory for a human domain expert, than the domain model used by the planning framework does not correspond well with the mental model of the expert, and needs therefore to be refined. In this sense, explanations are not aimed at a final user of the planning framework, but at an expert that is "optimising" the system before deployment, or for maintenance purposes.

Knowledge planning models play a central role in any planning application, and their importance has been well-argued (McCluskey, Vaquero, and Vallati 2017). During the Knowledge Engineering (KE) process of encoding specifications into an appropriate domain model, a number of decisions are usually taken by experts. Some of those decisions are due to the knowledge of the encoder, while others are due to common knowledge about the application "context". Such knowledge is usually not stored anywhere, but can provide valuable information for explaining both plans and the planning process to stakeholders. In this paper, we argue about the importance of context awareness and annotation of decisions made by domain model encoders for improving the quality of the planning and plans explanation. We categorise the extended set of questions that can be answered by exploiting such knowledge. Finally, we propose the overall architecture that would allow to capture relevant knowledge, and analyse it in order to provide suitable explanations.

Knowledge Engineering for Planning

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 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. 2008). However, the peculiarities of automated planning 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* workshop and the *Verification and Validation of Planning Systems* workshop, as well as competitions such as the *International Competition on Knowledge Engineering for Planning & Scheduling* (Chrpa et al. 2017). The competitions have motivated the development of powerful KEPS systems and advances in domain modelling techniques, languages and analysis approaches. Well-known examples of existing tools include GIPO (Simpson, Kitchin, and McCluskey 2007), itSimple (Vaquero et al. 2012), KEWI (Wickler, Chrpa, and McCluskey 2014), PDDL-studio (Plch et al. 2012), and most recently the `planning.domains` framework.

Enhancing the Knowledge Engineering Process to Support Explainability

In order to describe how explainability can be supported also by the KE process, let us consider a variant of the well known Blocksworld domain as a running example. The main difference from the well-known classical planning domain, in our example there are two types of objects, categorised according to their shape: cubes and cones. Figure 1 shows a possible PDDL encoding of two operators from the domain model, that allow to move objects from an initial cube to another one. The name of the other operators are listed, but their details are not given due to the lack of space.

In a real-world planning application, the domain model is usually not shown to the final user, also because the typical user does not have a strong background in automated planning, and is usually not interested in such details. It is therefore reasonable to assume that the user will reason, and require explanations, on the basis of provided solution plans. For this reason, on the basis of the operators shown in Figure 1, let us consider the following problem, described in terms of initial and goal states, and a (very short) solution plan. For the sake of readability, predicates corresponding to objects stacked on the same tower are shown on a single line. In the

```
(:action cube-to-cube
:parameters (?b1  ?b2  ?b3 - cube)
:condition (and  (on ?b1 ?b2) (clear ?b1) (clear ?b3))
:effect (and (not (on ?b1 ?b2)) (on ?b1 ?b3)
  (not (clear ?b3)) (clear ?b2) ))

(:action cone-to-cube
:parameters (?b1  ?b2 - cube ?b3 - cone)
:condition (and (on ?b3 ?b1) (clear ?b2) (clear ?b3))
:effect (and (not (on ?b3 ?b1)) (on ?b3 ?b2)
  (not (clear ?b2)) (clear ?b1) ))

(:action cone-to-table ... )
(:action cube-to-table ... )
(:action cone-from-table ... )
(:action cube-from-table ... )
```

Figure 1: A possible encoding of a variant of BlocksWorld, considering cube and cone blocks.

solution plan, the first number indicates the time step.

```
initial state:
    (on-table cube1) (on cone1 cube1) (clear cone1)
    (on-table cube2) (on cube3 cube2) (clear cube3)
    (on-table cube4) (clear cube4)
    (on-table cube5) (clear cube5)

goal:
    (on cube2 cube1)
    (on cube5 cube3)

plan:
    [0] (cone-to-table cone1 cube1)
    [1] (cube-to-cube cube3 cube2 cube4)
    [2] (table-to-cube cube2 cube1)
    [2] (table-to-cube cube5 cube3)
```

Remarkably, the Explainable planning framework proposed by Fox, Long, and Magazzeni (2017) is capable of effectively addressing requests of explanation about the behaviour of the planner that generated the plan. However, there are some categories of questions that can not be dealt with by considering only the planning models and the planning process. Let us examine such categories by considering the following examples.

- *Why is cube3 moved on top of cube4, and not on cone1?*
  A possible and trivial answer to the first question can be obtained by analysing the domain model: there is no operator that allows to move a cube over a cone. Clearly, this sort of responses are not particularly informative for the user, as they do not provide additional information on top of what is already available for the user to directly investigate. Instead, a child that is used to play with building blocks could provide a very intuitive yet informative explanation: a cube stacked on a cone will fall, so you do not even try to stack them. Similarly, the child would be happy to answer questions like *Why am I not able to move a block that is not clear?* To explain the plans and, to some

extent, the knowledge of a planning model, the child is implicitly exploiting his awareness of the domain, that is commonly termed as context awareness. In other words, the knowledge needed for answering that class of questions, is given by the application context. Hereinafter we will refer to this class of explanations as *context-related*.

- *Why (some) actions can be executed in parallel?*, *Why the exploited hand is not indicated?*

  As in the previous case, a trivial explanation would be that actions are executed in parallel when they are independent, that is, they do not have conflicting effects and their effects do not interfere with pre-conditions of the other actions. In order to provide an informative answer to this question, it is necessary to reason in terms of decisions and assumptions made during the modelling process, on the basis of the available requirements specification. In the specific case, for instance, requirements may specify that more than one "hand" is available to move blocks. For safety reasons, one hand is not allowed to deal, at a given time step, with blocks involved in tasks performed by another hand. On top of that, an expert knowledge engineer for automated planning can come up with a model like the one presented in Figure 1, where the robotic hands are not explicitly modelled, in light of the fact that –regardless of the number of available hands– actions to be performed at the same time step in the plan can be easily serialised and distributed among available hands. This class of explanations is therefore related to requirements and modelling decisions and assumptions. We refer to this class as *assumption-related*.

Having identified two classes of explanations that can be addressed by extending the knowledge encoded in the models, it is now crucial to understand from where, and how, such additional knowledge can be extracted.

Knowledge required for providing context-related explanations can be obtained by linking, during the KE process of the domain model, objects and entities with both common sense and context-specific knowledge ontologies. Such ontologies can semantically enhance notions by deriving context-related constraints and relationships. They are commonly applied in areas such as robotics or smart homes (Kunze, Tenorth, and Beetz 2010; Alirezaie et al. 2017), where they support the interaction between robots (or environment) and humans, by supporting the interpretation of commands and instructions. Remarkably, there exists approaches to semantically enrich PDDL problem models, in order to improve the reasoning capabilities of domain-independent planning engines (Dornhege et al. 2012). A similar approach, that supports knowledge share among domains with similar contexts, could be taken also on the KE side of planning, as explained in the next section.

Assumption-related explanations require domain and model-specific knowledge. Some explanations can be directly derived by the requirement specifications, assuming they have been encoded using a formal or semi-formal language. However, many explanations –as those previously listed– require also knowledge about the engineering process, particularly in terms of decisions taken, impact of such decisions, and assumptions made. Decisions and assumptions have to be documented during the KE process, and should be linked to the relevant part of the model, or of the specification.

Utilising a Knowledge Engineering Environment for Generating Explanations

A knowledge engineering environment has the purpose of helping domain and AI experts collect together and formulate application knowledge to be used in a variety of ways. KEWI is such tool for encoding domain knowledge usable by domain experts for collecting knowledge that could be used in automated planning. It was developed in the context of a real industrial process application [PlanSig 2015]. Though KEWI was aimed at gathering knowledge about actions and change, it features a domain expert-friendly set of tools specifically for those not familiar with AI planning, within a *domain ontology*. KEWI's use of the 'concept' is fundamental to its language, and is related to the idea of an object class. A concept is defined by it's super-class and the roles that constrain how instances of the concept may be related to instances of other concepts. There are three different types of role constraints: "part-of" relations for aggregation, properties which contain attribute-values, and general relations. Properties of objects give an object's attribute a value. Both properties and relations must have some functional information using the :min :max keywords. This denotes the type of relation/property such many-many, one-many etc

The domain ontology describes a collection of concepts populated by objects which must be plugged into a concept hierarchy, ensuring that explanations can exploit more abstract knowledge about them. Thus more information is available to be involved in an explanation, both in terms of tightening up what makes a valid state, and in terms of relating concepts and objects. The first example below is a KEWI representation of the Dockworkers domain [NauHallab 04], encoded by Gerhard Wickler [Plansig paper 2015]. This shows examples of concept hierarchies (e.g. pallet and container are members of the more abstract class stackable). It asserts that in any state, a crane must be at exactly one location denoted via the "(:min 1) (:max 1))" terms. This gives us two essential pieces of information - a location must exist for those objects, and this location must be unique. As another example, the KEWI ontology asserts that a "pile" is attached to a location, and it holds a pallet and a stack of containers. This knowledge is generally not explicit in a planning domain model since it is unlikely to be important in the planning function.

Returning to the Blockworld example, a part of ontology of our variant of BlocksWorld as captured by KEWI is depicted in Figure 3. It captures relations between classes of objects with constraints determining how many instances of the relation can be true in a state.

Both these examples illustrate how better explanations are available in this context: explanation can refer to more accurate used of relations and objects, and to semantically related information such as aggregation and concept relations.

```
(:domain dwr
  (:class agent)
  (:class crane
    (:super-class agent)
    (:role at (:min 1)(:max 1)(:class location))
    (:role holds (:max 1)(:class container)))
  (:class robot
    (:super-class agent)
    (:role loaded-with (:max 1)(:class container))
    (:property has-colour (:min 1)(:max 1)
       (:type colour)))
  (:class location
    (:role occupied-by (:max 1)(:class robot)))
  (:class stackable)
  (:class container
    (:super-class stackable)
    (:role on (:max 1)(:class stackable))
    (:role piled-on (:max 1)(:class pallet))
    (:property paint (:min 1)(:max 1)
       (:type colour)))
  (:class pallet
    (:super-class stackable)
    (:role at (:min 1)(:max 1)(:class location))
    (:role top (:min 1)(:max 1)(:class stackable)))
  (:property colour
    (:values ( red green blue )))
  (:relation adjacent
    (:arguments ((?loc1 location)(?loc2 location)
    )))
```

Figure 2: A part of ontology of the Dock-worker's domain in KEWI

```
(:class object
(:role on (:max 1) (:class cube)))
(:class cube (:super-class object))
(:class cone (:super-class object))
(:class hand
(:role holding (:max 1) (:class object)))
(:class table
(:role on-table (:max k) (:class object)))
```

Figure 3: A part of ontology of our variant of BlocksWorld generated by KEWI

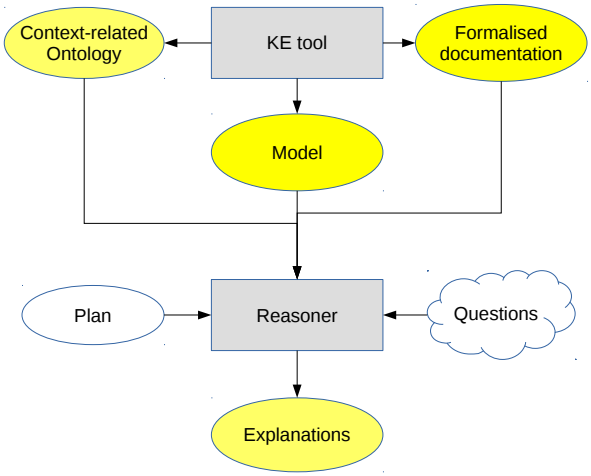

Figure 4: An overview of the components needed for supporting context-aware and assumption-aware explanations in planning.

Discussion

Three main approaches are traditionally exploited to formulate domain models.

1. Domain models are often formulated into a planner input language directly from a requirements specification using only (enriched) text editors (these are so-called *hand-crafted* domain models), such as the `planning.domains` framework.
2. Following a more principled process, requirements can be encoded firstly into a more application-oriented language such as UML in itSIMPLE, or in AIS-DDL in the KEWI interface, and then mapped into the target planner's input language. This can support a validation and verification processes, as well as improving maintenance and documentation of models.
3. Finally, there are approaches where automated acquisition tools can formulate partial or complete domain models. Tools, such as the LAMP system (Zhuo et al. 2010) and the LOCM system (Cresswell, McCluskey, and West 2013), can rely on different amount and type of available knowledge.

The first approach can hardly be modified to include the encoding of the additional knowledge that would be required in order to support context-related and assumption-related explanations. This is because the hand-crafting process is usually performed without producing significant documentation –sometimes even without any documentation at all– and is a try-and-fix process, where issues are tackled when they arise. This approach is supposed to be used for encoding mock-ups of domain models, in order to quickly testing some prototypes; however, evidence from recent ICKEPS indicate that this is usually not the case.

Tools designed for encoding models following the third approach, i.e. without humans in the loop, would also be hard to extend. The idea of such approaches is to generate models by considering available "raw" data, possibly collected from sensors, and to generate models that reflect the observed behaviour. The focus is not on human-readability, but on the encoding of models that can be immediately exploited for planning purposes. For this reason, it usually hard to link names and objects to the specific context. Moreover, assumptions are not made at the encoding level, but are implicitly given by the way in which tools have been designed and coded.

At this point, it should be clear that tools developed for encoding models following the depicted approach 2 are the best candidate to be extended for supporting explanations from the KE process. Tools like itSimple and KEWI come in the form of large frameworks that guide the encoding

of models. For this reason, they can be straightforwardly extended to support the level of documentation required for dealing with assumption-related explanations. Similarly to existing IDE for object-oriented programming, such KE tools can include plugins that facilitate documentation, and that underpin the semi-formalised encoding of comments, assumptions and motivations. This can be done by highlighting areas of the model that lack of "comments", and by posing *critical questions* (Atkinson and Bench-Capon 2007). Context-related explanations can be supported by appropriately designed plugins allowing to: (i) import an appropriate existing ontology, or design and fill a new ontology, (ii) link the ontology to the current KE process, and (iii) suggest, on the basis of the ontology, suitable object's types, predicates, etc. to be used. Aligning types and properties between the planning domain model and the ontology, allows to deal with context-related explanations.

Of course, extending the knowledge encoded during the KE process of a planning model is not enough to provide explanations. It is pivotal to provide a reasoner that is able to (i) analyse the questions, provided by the user or automatically generated, (ii) identify the most suitable source of knowledge, (iii) collect the available knowledge and perform automated reasoning, and (iv) provide one (or more) explanation addressing the question. Figure 4 shows an overview of an architecture capable of dealing with context- and assumption-related explanations. In yellow, we highlighted elements that can be a source of knowledge, or the product, of the explanation process.

Conclusion

It is pivotal that, in real-world planning applications, decisions and solutions can be explained, in order to maximise safety and increase reliability of the planning component.

In this paper we identified two categories of explanations that require additional knowledge, with regards to the knowledge provided in planning models and that can be extracted from the planning process, in order to be satisfyingly dealt with. Context-related explanations require information about the context that are not typically encoded in the planning models. Assumptions-related explanations are focused on the way in which models are encoded, and require information about decisions and assumptions made during the KE process.

We then described an architecture that would be able to deal with the introduced classes of explanations. Future work will focus on defining a suitable formalism for the additional knowledge, and in extending KE tools in order to effectively support the encoding of such knowledge during the KE process. Finally, we will investigate techniques and approaches –particularly from the argumentation community– that can be included in the automated reasoner shown in Figure 4.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; et al. 2004. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Alirezaie, M.; Renoux, J.; Köckemann, U.; Kristoffersson, A.; Karlsson, L.; Blomqvist, E.; Tsiftes, N.; Voigt, T.; and Loutfi, A. 2017. An ontology-based context-aware system for smart homes: E-care@ home. *Sensors* 17(7):1586.

Atkinson, K., and Bench-Capon, T. 2007. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence* 171(10-15):855–874.

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. 2008. Towards a structured approach to building qualitative reasoning models and simulations. *Ecological Informatics* 3(1):1 – 12.

Caminada, M. W. A.; Kutlák, R.; Oren, N.; and Vasconcelos, W. W. 2014. Scrutable plan enactment via argumentation and natural language generation. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, 1625–1626.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, 156–163.

Chrpa, L.; McCluskey, T. L.; Vallati, M.; and Vaquero, T. 2017. The fifth international competition on knowledge engineering for planning and scheduling: Summary and trends. *AI Magazine* 38(1):104–106.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using *LOCM*. *Knowledge Eng. Review* 28(2):195–213.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2012. Semantic attachments for domain-independent planning systems. In *Towards service robots for everyday environments*. 99–115.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable planning. *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*.

Kunze, L.; Tenorth, M.; and Beetz, M. 2010. Putting people's common sense into knowledge bases of household robots. In Dillmann, R.; Beyerer, J.; Hanebeck, U. D.; and Schultz, T., eds., *KI 2010: Advances in Artificial Intelligence*, 151–159.

McCluskey, T. L., and Vallati, M. 2017. Embedding automated planning within urban traffic management operations. In *Proceedings of the International Conference on Automated Planning and Scheduling ICAPS*.

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

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering knowledge for automated planning: Towards a notion of quality. In *Proceedings of the Knowledge Capture Conference, K-CAP*, 14:1–14:8.

Parkinson, S., and Longstaff, A. P. 2015. Multi-objective optimisation of machine tool error mapping using automated planning. *Expert Syst. Appl.* 42(6):3005–3015.

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.

Simpson, R.; Kitchin, D. E.; and McCluskey, T. 2007. Planning domain definition using gipo. *Knowledge Engineering Review* 22(2):117–134.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2011. Preferred explanations: Theory and generation via planning. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Vaquero, T. S.; Tonaco, R.; Costa, G.; Tonidandel, F.; Silva, J. R.; and Beck, J. C. 2012. itSIMPLE4.0: Enhancing the modeling experience of planning problems. In *System Demonstration – Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)*.

Wickler, G.; Chrpa, L.; and McCluskey, T. L. 2014. KEWI - A knowledge engineering tool for modelling AI planning tasks. In *KEOD 2014 - Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, 36–47.

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.