Acquisition and Re-use of Plan Evaluation Rationales on Post-Design

Tiago Stegun Vaquero[1] and **José Reinaldo Silva**[1] and **J. Christopher Beck**[2]
[1]Department of Mechatronics Engineering, University of São Paulo, Brazil
[2]Department of Mechanical & Industrial Engineering, University of Toronto, Canada
tiago.vaquero@usp.br, reinaldo@usp.br, jcb@mie.utoronto.ca

Abstract

In this article we investigate how knowledge acquired during a plan analysis phase can be represented, stored, and re-used to support the identification and evaluation of potential adjustments to a domain model. We describe a post-design framework that combines a knowledge engineering tool and an ontology-based reasoning system for the acquisition and re-use of human-centric rationales for plan evaluations. We aim at rationales that represent information about (1) why a given plan is classified as good or bad, (2) what are the properties of the plan that directly impact its quality, and (3) how these properties affect the plan quality, positively or negatively. This paper shows a case study, based on a benchmark problem, which illustrates the process of development and acquisition of rationales.

Introduction

Decisions about knowledge modeling and planning algorithm development drastically affect the quality of plans. From a planning technology perspective, in a *ceteris paribus* scenario, factors such as the improper choice of planning techniques and heuristics may lead to the generation of poor solutions. From a knowledge engineering perspective, lack of knowledge, ill-defined requirements and inappropriate definition of metrics, constraints and preferences can contribute directly to malformed models and, consequently, to unsatisfactory plans, independent of the planning algorithm. Traditionally, much of planning research has focused on a perspective in which new algorithms are developed and tuned to obtain high performance and better plans. Not much investigation has been done from the knowledge engineering (KE) perspective, especially re-modeling and refining the planning problem based on observations and information that emerge during the design process itself.

In plan analysis, hidden knowledge and requirements captured from human feedback raise the need for a continuous re-modeling process. The capture, representation and use of such human-centered feedback is still an unexplored area in the knowledge engineering for AI planning. Moreover, the impact of such feedback and re-modeling on the planning performance is unknown.

In (Vaquero, Silva, and Beck 2010), we propose of a post-design tool for AI planning that combines the open-source KE tool itSIMPLE (Vaquero et al. 2007) and a virtual prototyping environment called Blender to support the short-term identification of inconsistencies and hidden requirements. In that work, there is a first attempt to manually interpret and insert rationales to support design decisions in a model adaptation approach. However, the proposed tool did not provide a process to capture, store and evaluate rationales. Meanwhile, the discussion about the use of rationales has grown very fast specially in the software engineering community, guiding the process of documentation and reuse (Daughtry et al. 2009).

In this paper, we present a post-design tool for AI planning that addresses the acquisition and re-use of human-centered rationales for plan evaluations. We aim at rationales that describe the reasons behind the plan classification (e.g., bad or good quality) given by designers or users. Such rationales describe what are the properties of the plan that impact its quality and how these properties affect the plan quality: positively or negatively. We study an approach that combines the open-source KE tool itSIMPLE (Vaquero et al. 2007) and an ontology-based reasoning system to support the capture, representation and re-use of rationales during plan evaluation. The aim of rationale re-use is to present to the human planner the plan properties and elements that are likely to impact its quality. This information becomes a starting point for the evaluation of a newly generated plan. In addition, the tool contributes to the continuous discovery process of hidden requirements (e.g., constraints and preferences) that were not identified during the virtual prototyping phase but can be available from user evaluation and justification.

This paper is organized as follows. First, we discuss concepts in knowledge engineering for planning and their role in plan analysis and model adaptation – processes that are generally performed in a post-design phase. We then focus on the contribution of rationales to plan design and life cycle, that is, the capture, analysis and re-use of rationales. Next, we present a case study based on the benchmark planning problem called Gold Miner. Following there is a discussion of the results and some concluding remarks.

Knowledge Engineering and Post-Design

Requirements engineering (RE) and knowledge engineering (KE) principles have become important to the suc-

cess of the design and maintenance of real world planning applications (McCluskey 2002). While pure AI planning research focuses on developing reliable planners, KE for planning research focuses on the design process for creating reliable models of real domains (McCluskey 2002; Vaquero et al. 2007). A well-structured life cycle to guide design increases the chances of building an appropriate planning application while reducing possible costs of fixing errors in the future. A simple design life cycle is feasible for the development of small prototype systems, but fails to produce large, knowledge-intense applications that are reliable and maintainable (Studer, Benjamins, and Fensel 1998).

Research on KE for planning and scheduling has created tools and techniques to support the design process of planning domain models (Vaquero et al. 2009; Simpson 2007). However, given the natural incompleteness of the knowledge, practical experience in real applications such as space exploration (Jónsson 2009) has shown that, even with a disciplined process of design, requirements from different viewpoints (e.g. stakeholders, experts, users) still emerge after plan generation, analysis, revision and execution (Hatzi et al. 2010). For example, the identification of unsatisfactory solutions and unbalanced trade-offs among different quality metrics and criteria (Jónsson 2009; Rabideau, Engelhardt, and Chien 2000; Cesta et al. 2008) indicates a lack of understanding of requirements and preferences in the model. These hidden requirements raise the need for iterative re-modeling and tuning process. In some applications, finding an agreement or a pattern among emerging requirements is an arduous task (Jónsson 2009), making re-modeling a non-trivial process.

A fundamental step in the modeling cycle is the analysis of generated plans with respect to the requirements and quality metrics. Plan analysis naturally leads to feedback and the discovery of hidden requirements for refining the model. We call '*post-design analysis*' the process performed after plan generation, in which we have a base model and a set of planners that provide the solutions to be evaluated. In fact, literature on plan analysis has shown interesting tools and techniques for plan animation (McCluskey and Simpson 2006; Vaquero et al. 2007), visualization (e.g. Gantt charts), virtual prototyping (Vaquero, Silva, and Beck 2010), and plan querying and summarization (Myers 2006).

Unfortunately, visualization and simulation approaches, such as the virtual prototyping used in our previous work (Vaquero, Silva, and Beck 2010), can not assure that all missing knowledge will emerge, specially in real planning problems. In many real cases, user feedback and rationales are hard to understand and compile; they are captured in pieces over time, making patterns hard to be identified. Analyzing plans individually in such real cases will probably not correctly emphasize the hidden requirements; they must be captured, pieced together and recognized. The accumulation of data from plan evaluation and their respective rationales can serve as a foundation for the identification of domain knowledge that cannot clearly or easily be detected during the first plan analysis interactions with visualization techniques. In this work, we want to go a step further on such investigation of post-design. We focus on studying the capture, representation of human-centric feedback from plan evaluation, in the form of rationales, and the reuse of such rationales for further evaluations.

Rationales in Planning

In software engineering, a design rationale is essentially the explicit recording of the issues, alternatives, tradeoffs, decisions and justifications that were relevant to the elements in the design. Rationales can be used in a number of ways in the design of an artifact:

- to explore and evaluate the various design alternatives discussed during the design process.
- to determine the changes that are necessary to modify a design.
- to facilitate better communication among people who are involved in the design process.
- to assist in making decisions during the design process.
- in design verification, to check if the artifact/product reflects what the designers and the users actually wanted.
- to re-use past experiences and to avoid the same mistakes made in the previous design.

Requirements engineering research has already reported the importance of rationale-based approaches; they have provided improvements in quality and reduction in costly errors that outweigh the costs of capturing rationales (Ramaesh and Dhar 1994).

In planning literature, rationale has been generally referred to the "why a plan is the way it is", and to "the reason as to why the planning decisions were taken" (Polyank and Austin 1998). These rationales, usually called *plan rationales*, have been recognized as an important type of information (Wickler, Potter, and Tate 2006; Polyank and Austin 1998) that can influence not only the plan synthesis process but the whole life cycle of a plan. In such life cycle, plan rationales can be acquired and used in the plan synthesis process itself, or in plan analysis, evaluation, explanation, plan indexing and retrieval, failure recovery, and plan communication.

Most of the work on plan rationales focuses on capturing and using them to improve the plan generation. The existing approaches of capturing plan rationales are related to the identification of planning decisions made by the planners (e.g., rationales in the form of causality, dependency) that stem from the planning process itself (planning trace). As an example of plan rationale related to a planner decision might be "action A is chosen at the state S because it achieves goal g" or "because A's effects match an open condition of partial plan p". These planning decisions are usually analyzed and re-used for making further similar decisions. For example, the usefulness of storing plan rationales to help future planning has been demonstrated by several types of case-based planners (Upal and Elio 1999). The case-based approach proposes that each planning decision within a plan be annotated with a rationale for making that decision. In this case, the planners remember past planning solutions and failures so they can be re-used or avoided in the future. The

idea behind storing these types of rationales is that a previously made and retrieved planning decision will only be applied in the context of the current planning problem if the rationales for it also holds in the current problem (Upal and Elio 1999). The work of Wickler, Potter, and Tate (2006) describes a recording process of rationales into a plan ontology in which a planner can record the justifications for including components into the plan represented in the <I-N-C-A> ontology within the framework called I-X. In addition, research on plan rationales has also focused on learning and using such information to produce control knowledge or plan-refinement strategies, which would, as a result, improve the plan quality. A good review of plan rationales is provided in (Polyank and Austin 1998).

Design and decision rationales coming from people have received the least amount of attention in the planning literature. Differently from existing work on rationales in planning, we focus on acquiring human-centric rationales that emerge from user feedback, observations and justifications during plan evaluation. Based on general and individual criteria, interests, feelings and expectations, the rationales from plan evaluation generate explanations and justifications as to why a plan was classified into a specific quality level. Therefore, we extend here the concept of plan rationales with rationales that encompass "why a certain plan element does or does not satisfy a criterion" or "why a certain plan does or does not satisfy a preference". Moreover, these rationales could explain "why a certain metric does or does not satisfy a given criterion" and "what is the effect of a given plan characteristic or element in the plan quality" (e.g., it decreases or increases the quality). As an example of plan rationale, one might say that "the plan has a decreased quality because the robot left a block too close to the edge of the table" or "the plan has a high quality because the robot avoided repeatedly passing through the two most crowded areas of the building while cleaning it". We call these explanations *plan evaluation rationales*.

In this paper we focus on the implementation of the processes related to plan evaluation and the acquisition and re-use of rationales. We have designed a framework called *Post-Design Application Manager* (postDAM) that integrates itSIMPLE and a reasoning system called tuProlog[1] (a Java implementation of the Prolog engine). The implemented framework supports users on the following processes: classification of metrics and plans (plan evaluation), and the acquisition and re-use of rationales. In this paper we focus on the acquisition and re-use of rationales processes to support plan analysis.

Acquiring Rationales for Plan Evaluations

One of the main goals of this work is to capture the knowledge behind the classifications made upon the metric values and the plans. Rationales for plan evaluation may refer to elements and properties of the plan, including the plan structure itself. Therefore, it is necessary in the first place to consider a formal foundation of terms, concepts, relations and axioms to provide the base vocabulary of plan elements that can be used to specify a rationale. In this work, such a formal foundation is the Plan Ontology. Before introducing how rationales are captured and represented, we first describe the plan ontology utilized in the postDAM.

Representation of Evaluated Plans

As mentioned by Tate (Tate 1996), a richer plan representation could provide the following: a common basis for human and system communication about plans; a shared model of what constitutes a plan; mechanisms for automatic manipulation and analysis of plans; a target representation for reliable acquisition of plan information and feedback; formal reasoning about plans and re-use mechanisms. Such a representation is often based on an ontology - a plan ontology - which explicitly specifies the intended meanings of the terms being used, such as processes, activities, the constraints over their occurrences, or the meaning of the planning problem itself (Grüninger and Kopena 2005).

Among different ontologies for representing plans, we have chosen the Process Specification Language (PSL) (Schlenoff, Knutilla, and Ray 1996; Grüninger and Menzel 2003; Grüninger and Kopena 2005). PSL is an expressive ontological representation language of processes (plans), including activities and the constraints on their occurrences. PSL has been designed as a neutral interchange ontology to facilitate correct and complete exchange of process information among manufacturing systems such as scheduling, process modeling, process planning, production planning, simulation, project management, workflow, and business process applications (Grüninger and Kopena 2005). An interesting aspect of the PSL architecture is that it supports a set of extensions. A designer can extended PSL precisely to fit their expressive needs. We use PSL as the base for the plan ontology utilized in the postDAM framework.

Two of the most important terms in the core of PSL Ontology are the *activity* and its *occurrence*. As described in (Grüninger and Menzel 2003), an activity is a repeatable pattern of behavior, while an activity occurrence corresponds to a concrete instantiation of this pattern. For example, the term *pickup(r,x)* can denote the class of activities for picking up some object x with robot r, and the term *move(r,x,y)* can refer to the class of activities for moving robot r from location x to location y. The ground terms such as *pickup(Robot1,BlockA)* and *move(Robot1,LocA,LocB)* are instances of these classes of activities, and each instance can have different occurrences (e.g., two different occurrences of *move(Robot1,LocA,LocB)* can appear in plan). In fact, activities may have several or no occurrences. The relationship between activities and activity occurrences is represented by the *occurrence_of(o,a)* relation. Any activity occurrence corresponds to a unique activity. *Object* is also a term in PSL-core ontology. An *object* might be an argument of activities or fluents. Fluents are used to describe facts. For example, *at(Robot1,LocA)* is a fluent in PSL while *Robot1* and *LocA* are objects. Moreover, the term *State* is also used in PSL ontology, in its extensions. *States* may refer to a particular situation of the domain or to a fluent. Relationships such as *prior(f,o)* and *holds(f,o)* denote, respectively, a fluent (or state) f that holds prior to the activity occurrence o and af-

[1] tuProlog: see http://alice.unibo.it/xwiki/bin/view/Tuprolog/.

ter such occurrence. The complete lexicon of the language is detailed in the PSL website.[2]

As opposed to most applications of the PSL ontology, in this work we focus on the representation of plans generated by automated planners. Therefore, additional semantics and vocabularies must be considered. We have extended the PSL lexicon to include the terms, characteristics and vocabulary of AI planning, including *domain*, *problem*, *operators*, *pre- and post-conditions*, *objects*, *fluents that define states*, *plan*, *metrics*, and *plan quality*. Table 1 summarizes such additional terms and vocabulary used in the postDAM framework to represent a plan and its quality.

domain(d)	d is a domain
problem(i)	i is a problem instance
problem_of(i,d)	i is a problem instance of d
problem_solving_activity_of(a,i)	a the problem-solving activity of problem instance of i
plan(p)	p is a plan
solution_of(p,d,i)	p is a solution to the problem i of domain d
fluent_of(f,s)	f is a fluent of the state s. A set of fluents defines a state
numeric_fluent_of(f,v,s)	f is a numeric fluent with value v in the state s
positive_precondition(a,f)	f is a precondition of a
negative_precondition(a,f)	f is a negative precondition of a
effect(a,f)	f is a effect of a
negative_effect(a,f)	f is a negative effect of a
metric(m)	m is a metric
metric_value(p,m,v)	m has value v in plan p
metric_quality(p,m,q)	m has quality value q in plan p
quality(p,q)	p has quality value q

Table 1: Addition terms and relations to PSL Ontology used in postDAM

As an example of the ontological representation of plans in postDAM, let us suppose a simple planning problem from the classical blocks world domain in which block B must be unstacked from block A and put on the table. A plan with two actions solves the problem. In such example, the plan structure is represented along with the basic information about domain and problem, as well as the initial state. Figure 1 illustrates the two blocks example, along with basic concepts and elements of the proposed PSL extension (some of the terms are omitted in the figure to provide a clear view of ontology structure). The top area of Figure 1 illustrates the terms of the domain of application and the problem, whereas the bottom illustrates the plan structure.

Reasoning about a given plan would require the proper encoding of the PSL ontology, including the plan representation, action specifications and propagation rules. Such reasoning could be used to infer or check plan properties and characteristics (e.g. to infer in which state a given goal or fluent is achieved). We use Prolog for encoding the ontology in the postDAM framework.

[2]PSL is available at http://www.mel.nist.gov/psl/ontology.html.

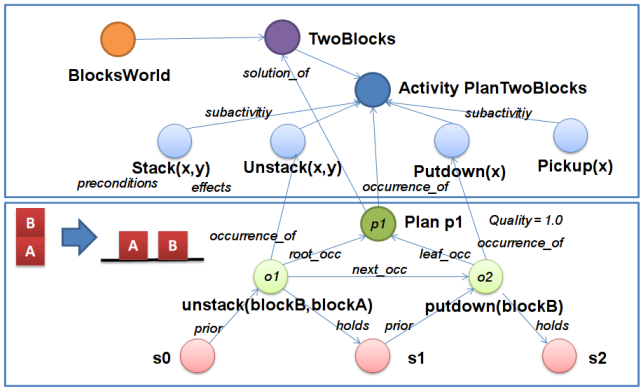

Figure 1: An illustration of the plan structure in the plan ontology used in postDAM

Representation of Rationales for Plan Evaluations

We aim at rationales that explain what affects the quality of plans (positively and negatively) and consequently why a plan has a given classification (including factual reasons or preferences). Conceptually, *plan evaluation rationales* in postDAM have the following straightforward format:

$$if < condition > then < effect\ on\ plan\ quality > \quad (1)$$

The *condition* of the rationale can be any logical sentence involving the properties of a plan, while the *effects* are predefined terms that specify whether the condition increases or decreases the quality of the plan. A rationale may refer to the effect on plan quality or on a particular metric quality. For example, one may have the following evaluation rationale: "if (*action1* occurs after *action2* AND *action3* is the last action in the plan) then (plan quality decreases)".

In order to represent rationales, we use the plan ontology described above along with a new vocabulary to capture the concepts of plan evaluation rationale. The vocabulary includes an important term called *rationale(r)* to represent an evaluation rationale. The relation *quality_rationale_of(r,p,j)* denotes the relationship between a rationale r and the plan p along with a user justification j. Depending on the ontology encoding, the justification might be a string object such as in the following example: *quality_rationale_of(r1,p1,"when action1 occurs after action2 and action3 is the last action in the plan the quality is decreased")*. The effects on plan quality are represented using the relations *affect_plan_quality(r,p,e)* and *affect_metric_quality(r,p,m,e)* where e can be either *increase* or *decrease*. The former refers to the direct effect on the quality of a plan, while the latter refers to the effect on a particular metric m. The following sentences represent the conceptual format of the rationale (sentence 1) using the above extended vocabulary:

$$< condition > \rightarrow quality\_rationale\_of(r, p, j).$$
$$quality\_rationale\_of(r, p, j) \rightarrow$$
$$affect\_plan\_quality(r, p, e).$$

In the first sentence, if *condition* is satisfied then relation *quality_rationale_of* becomes true, and consequently relation *affect_plan_quality* also becomes true according to the second sentence. The rationale's condition is represented using the vocabulary defined in the plan ontology.

We also consider different abstraction levels of rationales in the extended ontology. Rationales can be classified as *problem-dependent*, *domain-dependent* or *domain-independent*. Problem-dependent rationales are those that are only applied in the context of a particular problem instance (e.g., "if a robot moves to location 7 the plan has a low quality"). Domain-dependent rationales are those that are applicable in all problem instances of a particular domain (e.g., "if any robot performs a loop in its path the resulting plan quality is low"). Finally, domain-independent rationales are those applied to any planning domain or set of domains (e.g., "if a vehicle performs a loop of movements then the quality of the plan will be decreased"). The relation used to represent these concepts is the *abstraction_level(r,l)*, where l refers to one of the three levels of abstraction. The level of abstraction of rationales assists their re-use as will be discussed in th next section. Table 2 summarizes the terms and relations for representing rationales in postDAM.

rationale(r)	r is a rationale
quality_rationale_of(r,p,j)	r is a rationale of plan p with justification j
affect_plan_quality(r,p,e)	r describes the effect e on quality of p
affect_metric_quality(r,p,m,e)	r describes the effect e on quality of metric m of p
abstraction_level(r,l)	l is the abstraction level of r

Table 2: Rationales terms and relations in postDAM

The following examples represent two rationales using the vocabulary in Table 2.

$$rationale(r1).$$
$$abstraction\_level(r1, problem-dependent).$$
$$\forall p, o, sg \cdot leaf\_occ(o, p) \wedge holds(sg, o) \wedge fluent\_of(at(robot1, loc1), sg) \rightarrow quality\_rationale\_of(r1, p, j1).$$
$$\forall p \cdot quality\_rationale\_of(r1, p, j1) \rightarrow affect\_plan\_quality(r1, p, increase).$$

$$rationale(r2).$$
$$abstraction\_level(r2, problem-dependent).$$
$$\forall p, o, sg, v \cdot leaf\_occ(o, p) \wedge holds(sg, o) \wedge numeric\_fluent\_of(traveleddistance(robot1, v), sg) \wedge v > 50 \rightarrow quality\_rationale\_of(r2, p, j2).$$
$$\forall p \cdot quality\_rationale\_of(r2, p, j2) \rightarrow affect\_metric\_quality(r2, p, fuelused, decrease).$$

The above example represents two rationales, $r1$ and $r2$. The first rationale can be interpreted as follows: "The quality of a plan increases whenever the *robot1* is at *loc1* in the state generated by the last activity occurrence in the plan (*leaf_occ(o,p)*), and the goal state is still *sg*". The second rationale has the following meaning: "The quality of the metric *fuelused* decreases whenever the traveled distance of the *robot1* (variable *v*) is greater than 50 to the goal state". These rationales are manually inserted and maintained by users using itSIMPLE's interface.

Re-using Rationales for Plan Evaluation

During the plan evaluation, stored rationales can be re-used to support classification and justification. When a new plan is created, we can use past evaluations to identify the good and bad characteristics of the plan. In postDAM, the re-use process involves the tool itSIMPLE, the reasoning interface tuProlog and the *Plan Analysis Database*. The information from the abstraction level of rationales is essential to the re-use process. Supposing a new plan pn is generated to solve a problem instance i in the domain d, the process of re-using existing rationales is as follows:

1. itSIMPLE first translates the domain operators, the objects, the problem instance, and the new plan pn to the PSL plan ontology.
2. itSIMPLE accesses the plan analysis database to select the following rationales: (1) problem-dependent rationales that are applied specifically to the problem i; (2) domain-dependent rationales that are applied to any problem instance in domain d; and (3) domain-independent rationales that are applied to any planning domain.
3. The translated plan and selected rationales are put together with the PSL-Core and core theories forming a knowledge base to the plan.
4. The knowledge base is then read by the tuProlog for inference requests.
5. itSIMPLE accesses tuProlog to check which rationales are applied to plan pn. itSIMPLE requests the inference of *quality_rationale_of(R,p1,J)*, where R and J are variables to be instantiated by the Prolog engine.
6. The list of applied rationales (instantiated values of variable R), along with their respective justification (instantiation values of variable J), is read by itSIMPLE and attached to the new plan pn (in the XML format).

The re-used rationales can be seen in the evaluation summary provided by itSIMPLE.

Case Study

In this section, we present a case study using a benchmark domain from the *International Planning Competitions* (IPC) to evaluate the acquisition process of plan evaluation rationales in a post-design domain adaptation: the Gold Miner domain. This domain was chosen from a recent KE competition, because it is intuitive and has a clear correspondence between objects in the real and virtual world.

The procedure used for the case study is as follows:

1. We gather all rationales found and observations made during a virtual prototyping phase of plans generated by eight state-of-the-art planners: SGPlan5, MIPS-xxl 2006, LPG-td, MIPS-xxl 2008, SGPlan6, Metric-FF, LPG 1.2, and hspsp. For more details about the virtual prototyping phase see (Vaquero, Silva, and Beck 2010)
2. We represent the rationales in itSIMPLE using the proposed extension of the PSL ontology and the acquisition process described above.
3. We then analyze the rationales regarding their applicability, re-use and generality.

The Gold Miner Domain

The Gold Miner is a benchmark domain from the learning track of IPC-6 (2008). In this domain, a robot is in a mine and has the objective of reaching a location that contains gold. The mine is represented as a grid in which each cell contains either hard or soft rock. There is a special location where the robot can either pickup an unlimited supply of bombs or pickup a single laser cannon. The laser cannon can be used to destroy both hard and soft rock, whereas the bomb can only penetrate soft rock. If the laser is used to destroy a rock that is covering the gold, the gold will also be destroyed. However, a bomb will not destroy the gold, just the rock. This particular domain has a simple optimal strategy[3] in which the robot must (1) get the laser, (2) shoot through the rocks (either soft of hard) until it reaches a cell neighboring the gold, (3) go back to get a bomb, (4) explode the rock at the gold location, and (5) pickup the gold. In this case study we used the propositional typed PDDL model from the testing phase of IPC-6.

During the virtual prototyping and model refinement cycles performed with the Gold Miner domain described in (Vaquero, Silva, and Beck 2010), a set of observations were made. We summarize these observations as follows:

- One planner generated invalid solutions in which the robot used the laser at the gold location, destroying the gold (Vaquero, Silva, and Beck 2010).
- Some planners provided (valid) plans in which the laser cannon was fired at an already clear location.
- The laser cannon was left in a different position from the initial one. It would be better if the robot could leave the laser only at the same spot as the bomb source.
- Unnecessary move actions were present in some plans.

In this case study, we address each one of the above rationales in the postDAM framework.

The undesirable firing behavior of the laser cannon naturally decreases the quality of the plan, and specifically to the *laserusage* metric. The following rationale $rGM1$ denotes the issue of firing to an already clear position (the element $jGM1$ represents the user's justification of rationale $rGM1$):

$$
\begin{array}{l}
rationale(rGM1).\\
\forall p, o, t, x, y, s \cdot subactivity\_occurrence(o, p) \wedge \\
\quad occurrence\_of(o, firelaser(t, x, y)) \wedge \\
\quad prior(s, o) \wedge fluent\_of(clear(y), s) \rightarrow \\
\quad quality\_rationale\_of(rGM1, p, jGM1).\\
\forall p \cdot quality\_rationale\_of(rGM1, p, jGM1) \rightarrow \\
\quad affect\_plan\_quality(rGM1, p, decrease).
\end{array}
$$

The justification $jGM1$ could be encoded as a string like *jGM1 = 'Laser fired to nowhere (clear position), at ' + y*. Such a justification will be instantiated with every possibility of firing the laser cannon to a clear position y. itSIMPLE captures and records these instantiations in the XML representation of the plan as well as in the database. The evaluations summary provided by the tools shows such justifications to the users.

[3]IPC-6 2008. http://eecs.oregonstate.edu/ipc-learn/

The above rationale is applicable to plans of any problem instances in the Gold Miner domain and, therefore, can be considered in this work as a domain-dependent rationale ($abstraction\_level(rGM1, domain\text{-}dependent)$). However, one might consider it as a domain-independent rationale, applicable to a class of domains in which action $firelaser(t, x, y)$ exists and fluent $clear$ is used as one of the effects.

A specific undesirable firing behavior was also detected in some of the plans generated for the problem instance *gold-miner-target-5x5-01* from IPC. In this case, two laser cannon shots were made at rock locations that did not belong to a reasonable path to the gold position, consequently decreasing plan quality. The following rationale $rGM2$ represents such situation (the objects $node2i1$ and $node3i1$ represent the specific locations where the laser was fired):

$$
\begin{array}{l}
rationale(rGM2).\\
\forall p, o1, o2, t \cdot occurrence\_of(p, plangoldminertarget\text{-}5x501) \wedge subactivity\_occurrence(o1, p) \wedge \\
\quad subactivity\_occurrence(o2, p) \wedge \\
\quad occurrence\_of(o1, firelaser(t, node2i0, node2i1)) \wedge \\
\quad occurrence\_of(o2, firelaser(t, node3i0, node3i1)) \rightarrow \\
\quad quality\_rationale\_of(rGM2, p, jGM2).\\
\forall p \cdot quality\_rationale\_of(rGM2, p, jGM2) \rightarrow \\
\quad affect\_plan\_quality(rGM2, p, decrease).
\end{array}
$$

The condition of the rationale $rGM2$ checks the existence of two specific firing occurrences. Note that rationale $rGM2$ is restricted to solutions of the problem *gold-miner-target-5x5-01* by the condition $occurrence\_of(p, plangoldminertarget5x501)$ (where *plangoldminertarget5x501* is the activity of solving problem *gold-miner-target-5x5-01*). Therefore, this is a problem-dependent rationale (*abstraction_level (rGM2,problem-dependent)*).

During the virtual prototyping phase, we raised the issue of the position in which the laser cannon was left at the goal state. Leaving the cannon at the same position as the bomb source was preferred. The following rationale $rGM3$ denotes such a preference and expectation:

$$
\begin{array}{l}
rationale(rGM3).\\
\forall p, o, sg, x \cdot leaf\_occ(o, p) \wedge holds(sg, o) \wedge \\
\quad fluent\_of(bombat(x), sg) \wedge \\
\quad fluent\_of(laserat(x), sg) \rightarrow \\
\quad quality\_rationale\_of(rGM3, p, jGM3).\\
\forall p \cdot quality\_rationale\_of(rGM3, p, jGM3) \rightarrow \\
\quad affect\_plan\_quality(rGM3, p, increase).
\end{array}
$$

The condition of $rGM3$ checks the existence of both fluents $bombat(x)$ and $laserat(x)$ (where x is a location) at the goal state sg. In this case study, the rationale $rGM3$ is considered domain-dependent since it is applicable for all synthesized solutions for the Gold Miner domain.

In all refined models resulted from the virtual prototyping phase, unnecessary move actions appear in their respective plans. During the experiment, the rationales for detecting and explaining such undesirable characteristic evolve from a specific approach to a more general one (reaching a reusable representation). Due to limited space, we will focus on an example of rationale that referred to some loops in the

robot's path. This rationale captured during the experiment has the following representation:

$rationale(rGM4)$.
$\forall p, o1, o2, a, t, x1, x2 \cdot subactivity\_occurrence(o1, p) \wedge$
$subactivity\_occurrence(o2, p) \wedge$
$next\_subocc(o1, o2, a) \wedge$
$occurrence\_of(o1, move(t, x1, x2)) \wedge$
$occurrence\_of(o2, move(t, x2, x1)) \rightarrow$
$quality\_rationale\_of(rGM4, p, jGM4)$.
$\forall p \cdot quality\_rationale\_of(rGM4, p, jGM4) \rightarrow$
$affect\_plan\_quality(rGM4, p, decrease)$.

Rationale $rGM4$ refers to all sequences of move actions that take a robot from location $x1$ to $x2$ and then back to $x1$ immediately after that. As the re-modeling process progressed, more sophisticated explanations can be constructed manually (Vaquero 2011). For example, a rationale could be defined to detect any complex loop ($x1$, $x2$, ... , xn, $x1$) performed by a robot. The following rationale representation captures any outer loop in a single robot's path:

$rationale(rGM5)$.
$\forall p, o1, o2, x, y, z, a \cdot subactivity\_occurrence(o1, p) \wedge$
$subactivity\_occurrence(o2, p) \wedge$
$occurrence\_of(o1, move(t, x, y)) \wedge$
$occurrence\_of(o2, move(t, y, z)) \wedge$
$next\_subocc(o1, o2, a) \rightarrow$
$consecutive\_move(o1, o2, t, p)$.
$\forall p, o1, o2, o3, x, y, z, w, h, a \cdot$
$subactivity\_occurrence(o1, p) \wedge$
$subactivity\_occurrence(o2, p) \wedge$
$subactivity\_occurrence(o3, p) \wedge$
$occurrence\_of(o1, move(t, x, y)) \wedge$
$occurrence\_of(o2, move(t, z, w)) \wedge$
$occurrence\_of(o3, move(t, h, z)) \wedge$
$min\_precedes(o1, o2, a) \wedge next\_subocc(o3, o2, a) \wedge$
$consecutive\_move(o1, o3, t, p) \rightarrow$
$consecutive\_move(o1, o2, t, p)$.
$\forall p, o1, o2, x, y, z, a \cdot subactivity\_occurrence(o1, p) \wedge$
$subactivity\_occurrence(o2, p) \wedge$
$occurrence\_of(o1, move(t, x, y)) \wedge$
$occurrence\_of(o2, move(t, z, x)) \wedge$
$min\_precedes(o1, o2, a) \wedge$
$consecutive\_move(o1, o2, t, p) \rightarrow$
$loop\_of\_move(o1, o2, t, p)$.
$\forall p, o1, o11, o2, o22, x, y, l, h, a \cdot$
$occurrence\_of(o1, move(t, x, l)) \wedge$
$occurrence\_of(o2, move(t, h, x)) \wedge$
$min\_precedes(o1, o2, a) \wedge next\_subocc(o11, o1, a) \wedge$
$next\_subocc(o2, o22, a) \wedge$
$occurrence\_of(o11, move(t, y, x)) \wedge$
$occurrence\_of(o22, move(t, x, y) \rightarrow$
$has\_previous\_loop\_move(o1, o2, t)$.
$\forall p, o1, o2, t, x, y, z \cdot subactivity\_occurrence(o1, p) \wedge$
$subactivity\_occurrence(o2, p) \wedge$
$occurrence\_of(o1, move(t, x, y)) \wedge$
$occurrence\_of(o2, move(t, z, x)) \wedge$
$\neg has\_previous\_loop\_move(o1, o2, t) \wedge$
$loop\_of\_move(o1, o2, t, p) \rightarrow$
$quality\_rationale\_of(rGM5, p, jGM5)$.
$\forall p \cdot quality\_rationale\_of(rGM5, p, jGM5) \rightarrow$
$affect\_plan\_quality(rGM5, p, decrease)$.

In the above rationale, we define new relations to assist the identification of outer loops (for this context only) such as *consecutive_move(o1,o2,t,p)*, *loop_of_move(o1,o2, t,p)* and *has_previous_loop_move(o1,o2,3)*. The relation *consecutive_move(o1,o2,t,p)* and *loop_of_move(o1,o2,t,p)* together capture any consecutive sequence of move occurrences that constitutes a loop, whereas relation *has_previous_loop_move(o1,o2,t)* filters the outer loops. It is indeed possible to capture the existing inner loops performed by the robot, but we see them as redundant information when we capture the outer ones. The interesting point here is that we can design and create any supporting concept, relations and axiom for defining a rationale condition.

The rationales represented in this case study can be reused and applied in a new plan evaluation process. Using the integration between itSIMPLE and the reasoning system (tuProlog) we can automatically generate some justifications to the initial plan evaluation made by itSIMPLE. All justifications are based on the rationales acquired and inserted in the *Plan analysis Database* in this case study. For example, if a plan has a loop in the robot's path, the framework will be able to detect such a loop and pinpoint where it is happening to the user, as well as how it is affecting the plan quality (in this case negatively).

Discussion

The proposed acquisition process of human-centered rationales has shown to be feasible using an ontological approach. The case study provides an indication that these rationales can be an important source of essential knowledge such as user preferences. The reuse of past evaluation experience gives an important starting point in any analysis of a newly generated plan. Such re-use not only supports the knowledge acquisition process but also the decision process on already built applications. In the latter case, rationales can be applied to identify the trade-offs of plans selected for execution, knowing beforehand their advantages and disadvantages.

The case study has also shown that rationales can evolve during design and over time. As designer becomes familiar with the observations and users' intention, the representation of the collected rationales becomes more accurate, encompassing the exact envisaged situations. Therefore, rationales have their own maturity process in the design cycle.

Although we have not implemented the cross-project reuse of rationales in this work, it is possible to infer from the case study some examples about what could be enhanced in planning design patterns. In the Gold Miner domain experiment, the rationale related to the undesirable movement loops is a potential candidate for being attached to design patterns such as the *transportation* described in (Long and Fox 2000). Moreover, metrics utilized in the experiments could enhance the design patterns; some of them can be used in a number of planning applications such as the travel distance (a common quality measure in transportation and navigation benchmark domains).

We believe that the matching process between past evaluations and design patterns can speed up the design process itself, improve quality, reduce cost, and decrease problem fixing issues in real planning applications.

Conclusion

We have described a post-design framework that integrates a number of tools to assist the discovery of missing requirements, to support evaluation rationale acquisition and re-use, and to guide the model refinement cycle. In previous work we demonstrated that following a careful post-design analysis, we can improve not only plan quality but also solvability and planner speed (Vaquero, Silva, and Beck 2010). In this paper we demonstrated how evaluation rationales can be captured, represented and re-used. We discuss that this type of human-centric feedback can be useful and reusable in further plan evaluations and in other planning domains. In a real planning application, the analysis process that follows design becomes essential to have the necessary knowledge represented in the model.

References

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2008. Validation and verification issues in a timeline-based planning system. In *Proceedings of the ICAPS 2008 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Daughtry, J. M.; Burge, J. E.; Carroll, J. M.; and Potts, C. 2009. Creativity and rationale in software design. *ACM SIGSOFT Software Engineering Notes* 34:27–29.

Grüninger, M., and Kopena, J. B. 2005. Planning and the Process Specification Language. In *Proceedings of the ICAPS 2005 Workshop on the Role of Ontologies in Planning and Scheduling*, 22–29.

Grüninger, M., and Menzel, C. 2003. The Process Specification Language (PSL) theory and applications. *AI Magazine* 24(3):63–74.

Hatzi, O.; Vrakas, D.; Bassiliades, N.; Anagnostopoulos, D.; and Vlahavas, I. P. 2010. A visual programming system for automated problem solving. *Expert Systems With Applications* 37:4611–4625.

Jónsson, A. K. 2009. Practical Planning. In *ICAPS 2009 Practical Planning & Scheduling Tutorial*.

Long, D., and Fox, M. 2000. Automatic Synthesis and use of Generic Types in Planning. In *In Artificial Intelligence Planning and Scheduling AIPS-00*, 196–205. Breckenridge, CO: AAAI Press.

McCluskey, T. L., and Simpson, R. M. 2006. Tool support for planning and plan analysis within domains embodying continuous change. In *Proceedings of ICAPS 2006 Workshop on Plan Analysis and Management*.

McCluskey, T. L. 2002. Knowledge Engineering: Issues for the AI Planning Community. *Proceedings of the AIPS-2002 Workshop on Knowledge Engineering Tools and Techniques for AI Planning. Toulouse, France* 1–4.

Myers, K. L. 2006. Metatheoretic Plan Summarization and Comparison. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*. Cumbria, UK: AAAI Press.

Polyank, S., and Austin, T. 1998. Rationale in Planning: Causality, Dependencies and Decisions. *Knowledge Engineering Review* 13(3):247–262.

Rabideau, G.; Engelhardt, B.; and Chien, S. 2000. Using generic preferences to incrementally improve plan quality. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*. Breckenridge, CO.: AAAI Press.

Ramaesh, B., and Dhar, V. 1994. Representing and maintaining process knowledge for large-scale systems development. *IEEE Expert: Intelligent Systems and Their Applications* 9(2):54–59.

Schlenoff, C.; Knutilla, A.; and Ray, S. 1996. Unified process specification language: Functional requirements for modeling processes. In *National Institute of Standards and Technology*.

Simpson, R. M. 2007. Structural Domain Definition using GIPO IV. In *Proceedings of the Second International Competition on Knowledge Engineering for Planning and Scheduling*.

Studer, R.; Benjamins, V. R.; and Fensel, D. 1998. Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering* 25(1-2):161–197.

Tate, A. 1996. Representing plans as a set of constraints - the I-N-OVA model. In *Proceedings Third International Conference on AI Planning Systems (AIPS-96)*. Edinburgh: AAAI Press.

Upal, M. A., and Elio, R. 1999. Learning rationales to generate high quality plans. In *Proceedings of the Twelfth International FLAIRS Conference*, 371–377. Menlo Park, CA, USA: AAAI Press.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE2.0: An integrated Tool for Designing Planning Environments. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*. Providence, Rhode Island, USA: AAAI Press.

Vaquero, T. S.; Silva, J. R.; Ferreira, M.; Tonidandel, F.; and Beck, J. C. 2009. From Requirements and Analysis to PDDL in itSIMPLE3.0. In *Proceedings of the Third International Competition on Knowledge Engineering for Planning and Scheduling, ICAPS 2009*, 54–61.

Vaquero, T. S.; Silva, J. R.; and Beck, J. C. 2010. Improving Planning Performance Through Post-Design Analysis. In *Proceedings of the ICAPS 2010 Workshop on Scheduling and Knowledge Engineering for Planning and Scheduling (KEPS)*, 45–52.

Vaquero, T. S. 2011. *Post-Design Analysis for AI Planning Applications*. Ph.D. Dissertation, Polytechnic School of the University of São Paulo, Brazil.

Wickler, G.; Potter, S.; and Tate, A. 2006. Recording Rationale in <I-N-C-A> for Plan Analysis. In *Proceedings of the ICAPS 2006 Workshop on Plan Analysis and Management*.