Exploring Knowledge Engineering Strategies in Designing and Modelling a Road Traffic Accident Management Domain

Mohammad M. Shah, Lukáš Chrpa,
Diane Kitchin, Thomas L. McCluskey and **Mauro Vallati**
Department of Informatics
School of Computing and Engineering
University of Huddersfield, United Kingdom
Email:{s.shah, l.chrpa, d.kitchin, t.l.mccluskey, m.vallati}@hud.ac.uk

Abstract

Formulating knowledge for use in AI Planning engines is currently something of an ad-hoc process, where the skills of knowledge engineers and the tools they use may significantly influence the quality of the resulting planning application. There is little in the way of guidelines or standard procedures, however, for knowledge engineers to use when formulating knowledge into planning domain languages such as PDDL. This paper seeks to investigate this process using as a case study a road traffic accident management domain.

Managing road accidents requires systematic, sound planning and coordination of resources to improve outcomes for accident victims. We have derived a set of requirements in consultation with stakeholders for the resource coordination part of managing accidents. We evaluate two separate knowledge engineering strategies for encoding the resulting planning domain from the set of requirements: (a) the traditional method of PDDL experts and text editor, and (b) a leading planning GUI with built in UML modelling tools.

These strategies are evaluated using process and product metrics, where the domain model (the product) was tested extensively with a range of planning engines. The results give insights into the strengths and weaknesses of the approaches, highlight lessons learned regarding knowledge encoding, and point to important lines of research for knowledge engineering for planning.

1 Introduction

Knowledge Engineering for automated planning is the process that deals with the acquisition, formulation, validation and maintenance of planning knowledge, where a key product is the *domain model*. The field has advanced steadily in recent years, helped by a series of international competitions[1], the build up of experience from planning applications, along with well developed support environments (for example, *Europa* [Barreiro *et al.*, 2012], itSIMPLE [Vaquero *et al.*, 2007], GIPO [Simpson *et al.*, 2007]). It is generally accepted that effective tool support is required to build domain models and bind them with planning engines into applications. There have been reviews of such knowledge engineering tools and techniques for AI Planning [Vaquero *et al.*, 2011], and some work was done in comparing tools using sets of "features" for the ICKEPS competitions [Barták *et al.*, 2010]. While these works are illuminating, they are not founded on practice-based evaluation, in part, no doubt, because of the difficulty in setting up evaluations of methods themselves. Given a new planning domain, there is little published research to inform engineers on which method and tools to use in order to effectively *engineer a planning domain model*. This is of growing importance, as domain independent planning engines are now being used in a wide range of applications, with the consequence that operational problem encodings and domain models have to be developed in a standard language such as PDDL. In particular, at the difficult stage of domain knowledge formulation, changing a statement of the requirements into something formal - a PDDL domain model - is still somewhat of a "black art", usually conducted by a team of AI experts using text editors. On the other hand, the use of tools such as itSIMPLE or GIPO, with which experts generate a high level diagrammatic description and *automatically* generate the domain model, have not yet been proven to be more effective than hand coding.

In this paper we explore the deployment of automated planning to assist the management of accident planning, using a set of requirements derived from operational manuals and stakeholder interaction. Moreover, in introducing a new planning domain, we take the opportunity to employ and hence evaluate two separate methods for knowledge formulation: (i) the traditional method of hand-coding by PDDL experts, using a text editor and relying on dynamic testing for debugging; (ii) itSIMPLE [Vaquero *et al.*, 2007], an award-winning GUI, utilising a method and tool support based on the Uni-

[1] for the most recent see http://icaps12.poli.usp.br/icaps12/ickeps

fied Modelling Language (UML). Evaluating these two approaches with respect to qualitative and quantitative measures, gives a range of interesting insights into their strengths and weaknesses for encoding new domains. Evaluation measures used are based on two standard categories in the software engineering literature - process (the method of encoding and debugging the domain model) and product (the domain model, and its use within a planner to produce plans). In particular, we provide a comparison of the operationality of the planning domain models generated through the proposed methods, based on the performance of state-of-the-art domain independent planners.

2 Case Study: Management of Road Traffic Accidents (RTAs)

Road traffic management operations are subject to rising costs, rising public expectations, more complex and demanding goals, and contain a great deal of legacy software. Recent technological advances have in part confounded this by providing more management controls and more surveillance data. The need to look to reducing costs, while maintaining level of service is a high priority. The area of *incident management* on the Road Traffic Network combines the challenge above, while demanding an optimal solution in real time; further, the space of possible states, and configurations of the incident, is far too large to be able to generate concrete plans a priori. Systematic, robust planning with the coordination of human and technical resources is the key to managing these incidents. In particular where the process involves safety of victims and other road users, such as in RTA, the responding agencies need to deliver accident management activities safely and efficiently [Owens *et al.*, 2000].

Our work in producing a set of requirements for the automation of accident management plans has been performed in the context of the EU-funded network *Autonomic Road Transport Support*[2] consisting of both academics and practising transportation engineers. Using contacts through this network, two scientific exchanges with transportation specialists, contributions to transport workshops, and a set of manuals [HA, 2009; Owens *et al.*, 2000; Benesch, 2011], we have elicited a set of requirements for the RTA planning problem.

In the UK, the main responsibility for managing and dealing with an incident lies with the highways agency (HA) that serves that area, as well as the police, ambulance, traffic offers and breakdown services. Part 7 of the UK's Highway's Agency Manual [HA, 2009] is our major source of knowledge. This identifies the service providers responsible for dealing with accidents at an operational level, with police leading co-ordination in and around the scene. The phases of an incident are detection, verification, response, scene management, recovery and restoration. Here we assume that an accident has been detected, and consider the planning element for the subsequent phases, with the overall requirement that the planning function is to provide whoever is leading the incident management with an operational plan for managing services. Incidents are centrally controlled, and there is only one leader at any point in time (though leadership can change, e.g. from the police to the HA). Within this context there are *major* and *critical* levels of incident. The former can be described as disasters; the HA require a Crisis management Team to deal with this. We will concentrate on incidents at critical levels, which consist of single or multiple accidents in a region, and typically may consist of up to 10s of vehicles, requiring several emergency vehicles, within a single region.

[2] www.cost-arts.org

2.1 Initial Domain Analysis

An initial conceptualisation of the RTA domain is described in the following paragraphs.

A *Road Network* is represented by an undirected graph (V, E) where vertices V stand for *locations* and edges E for *roads*. It is useful to effectively abstract the topology of the Road Network, since the Road Network usually considers a region covering several 'clusters', i.e., towns/cities or districts (e.g. see Figure 1), with locations of interest (e.g. Hospitals or Police Stations). We assume that all the locations within a 'cluster' are connected to each other. 'Clusters' are connected only if there is a road between them. *Assets* $X = X_s \cup X_m$ are divided into two categories, *static assets* X_s (e.g. Police Stations, Hospitals, Fire Stations) and *mobile assets* X_m (e.g. Police Cars, Ambulances, Fire Brigades). Let $T \subseteq \mathbb{R}_0^+$ be a set of time-stamps. We define a function loc which for an asset and time-stamp returns the location (or $\perp$ which stands for a situation when the asset is on the way), formally $loc : X \times T \to V \cup \{\perp\}$. Clearly, for every static asset $x \in X_s$ $loc(x, t)$ is constant (i.e. its value is not dependent on the time-stamp). Mobile assets can be moved between locations using roads (i.e. a mobile asset can move from one location to another if and only if these locations are connected by road). Artefacts Y (e.g. accident victims, damaged cars etc.) cannot move freely between locations (unlike mobile assets) but they need a mobile asset (e.g. an ambulance) which can transport them to different locations. We define a function in which for an artefact and time-stamp returns either an asset (static or mobile) an artefact is attached to, or a location an artefact is located if the artefact is not attached to any asset, or $\perp$ if an artefact is being attached or detached from an asset, formally $in : Y \times T \to X \cup V \cup \{\perp\}$. An artefact can be attached to an asset if and only if the artefact is currently not attached to any other asset and the current location of an artefact is the same as the current location of the asset. Similarly, if an artefact is unattached from an asset then its location will be the same as the current location of the asset. Each asset may have a limited *capacity*, i.e., a maximum number of attached artefacts in the same time. We define a function $cap : X \to \mathbb{N}$ referring to an asset capacity. It must hold that $\forall t \in T, \forall x \in X : |\{y \mid y \in Y \wedge in(y, t) = x\}| \leq cap(x)$. Assets and artefacts can also interact with each other in order to modify their characteristic properties. For instance, the police have to confirm an accident or a paramedic has to give first aid to victims before they are taken to hospital. Hence, we define *properties* as sets of values characterising artefacts and/or assets (e.g. accident victims can be waiting for first aid, being aided, aided or delivered to a hospital).

All the above thus specify the environment of the RTA domain. This environment can be modified by (planning)

operators representing types of actions, specified via preconditions (what must be met in order to apply the operator) and effects (what is changed in the environment after applying the operator). We define the following operator families which modify the environment of the RTA domain (we assume that the operator is applied in a time-stamp t and lasts for Δt time).

move(x, l_1, l_2) moves a mobile asset $x \in X_s$ from a location l_1 to a location l_2 ($l_1, l_2 \in V$). As a precondition it must hold that $loc(x,t) = l_1$ and l_1 and l_2 are connected with a road (i.e. $(l_1, l_2) \in E$). An effect of applying the operator is that $loc(x, t + \Delta t) = l_2$ and $\forall t' \in (t, t + \Delta t) : loc(x, t') = \perp$.

attach(y, x, l) attaches an artefact $y \in Y$ to an asset $x \in X$ in a location $l \in V$. As a precondition it must hold that $loc(x,t) = in(y,t) = l, \forall t' \in [t, t+\Delta t] : loc(x,t') = l$ and $|\{y' \mid in(y',t) = x\}| < cap(x)$. An effect of applying the operator is that $in(y, t + \Delta t) = x$, $\forall t' \in (t, t + \Delta t) : in(y, t') = \perp$.

detach(y, x, l) detaches an artefact $y \in Y$ from an asset $x \in X$ in a location $l \in V$. As a precondition it must hold that $\forall t' \in [t, t + \Delta t] : loc(x, t') = l$ and $in(y,t) = x$. An effect of applying the operator is that $in(y, t+\Delta t) = l, \forall t' \in (t, t + \Delta t) : in(y, t') = \perp$.

interact$(e_1, e_2, l, p_1, \ldots, p_6)$ refers to interaction between artefacts or assets $e_1, e_2 \in X \cup Y$ in a location $l \in V$. As a precondition it must hold that $\forall t' \in [t, t + \Delta t] : (loc(e_1, t') = l \Leftrightarrow e_1 \in X) \vee (in(e_1, t') = l \Leftrightarrow e_1 \in Y)$, $(loc(e_2, t') = l \Leftrightarrow e_2 \in X) \vee (in(e_2, t') = l \Leftrightarrow e_2 \in Y)$ and e_1 and e_2 has properties p_1 and p_2 in time t. An effect of applying the operator is that properties of e_1 and e_2 in any $t' \in (t, t + \Delta t)$ are p_3, p_4 respectively and properties of e_1 and e_2 in $t + \Delta t$ are p_5, p_6 respectively.

There are further constraints which must be met. Operator families *attach* and *detach* must not be executed simultaneously for a given asset (i.e., during attaching or detaching an artefact no other artefact can be attached to or detached from the given asset). Also artefacts must have certain properties in order to be attached or detached from the assets (e.g. an accident victim must be stabilised before it is loaded to an ambulance).

Despite a very general scope of the operators' definitions it can illustrate well the main aspects of the RTA domain. Clearly, we may have to consider a 'cluster'-like topology of the Road Network by introducing two 'move' operators, one for moving within a 'cluster' and the other for moving between different 'clusters'. 'Attaching' and 'detaching' artefacts to/from assets must reflect different kinds of artefacts or assets. For instance, accident victims can be 'attached' to ambulances or hospitals, in other words the victim is loaded into the ambulance or is delivered into the hospital. 'Interacting' between assets and/or artefacts captures situations such as giving first aid to accident victims (an ambulance must be at the accident scene), certifying an accident by police (a police car must be at the accident scene), or untrapping accident victims by a fire brigade.

The RTA domain deals with a situation which arises immediately after a traffic accident has been reported. Police must certify and secure the accident scene. Fire brigades must free accident victims trapped in a vehicle, and fire brigades must extinguish any fire at the accident scene. Once victims are released and free of wreckage, paramedics must give first aid to them, and then load them into ambulances and deliver them to hospitals. Tow trucks then deliver damaged cars from the accident scene to garages.

3 The Knowledge Engineering Methods

Having analysed the RTA domain, and conceptualised the requirements, we set up the experiment to evaluate two methods of planning domain knowledge formulation. As with any exercise on evaluating methods, there are problems to do with the effect of human factors such as level of method expertise of the knowledge engineers (so-called extraneous variables). Each method was carried out in parallel over a period of time. There was no time limit fixed a priori. Each team was composed of two experts. All the experts were involved in the requirements phase. The background of all participating in the teams was that all except one (who is in the final year of an AI Planning PhD) had a PhD in AI Planning. The expertise level of the PDDL encoders (method A) was expert, whereas for method B the team leader was competent rather than expert in the use of the itSIMPLE tool.

The aim of the experiment was to evaluate two well known approaches to formulate domain models and problem encodings, within the RTA domain, where the problem was to generate management plans for a particular scenario. The criteria for evaluation are arranged in two broad categories, using inspiration from the software engineering area:

- the process of the formulation: this is the encoding of the conceptualised knowledge taken from the initial domain analysis, source documents and expertise, until it reaches a final form in which it can be input to AI Planning engine(s). Features such as defect identification and removal, the nature of testing, and repeatability/traceability of the process are considered here.
- the product of the formulation: this is the domain model and problem files, with features such as maintainability, size, complexity and operationality (the latter based on the performance and quality of plans produced and range of problems solved where they are consequences of the model rather than the planner).

As an application scenario, we have chosen a region shown in Figure 1, consisting of an area within the UK. For evaluating the methods we used a set of test instances, considering the map shown in Figure 1, in which three accidents happen in Ainley Top, Greetland and Baliff Bridge. Police are required to confirm accidents, number of victims and vehicles involved. The victims are required to be taken as soon as possible to one of the hospitals, and involved broken vehicles are required to be removed from the road and taken to an available garage. Three ambulances, four police cars, two fire brigades and four tow trucks are available. We created *test instances* involving from six to one hundred victims, and from

five to thirty cars, where some victims might be trapped inside cars. We also considered an instance in which the number of available emergency vehicles is doubled, to evaluate the coordination that the different methods encodings are able to achieve. In each case, the formulation proceeded until the method produces a planning application which solves the test instances of accident management as specified above.

We overview each method before we use it, but given space restrictions the reader is encouraged to consult the literature for full details.

3.1 Method A: Hand Encoding

PDDL [McDermott, 1998; Ghallab *et al.*, 1998] is an action-based domain definition language which is inspired by STRIPS [Fikes and Nilsson, 1971] style planning. The core of the PDDL formalism is for expressing the semantics of domain actions, using pre- and post- conditions to describe the applicability and effects of actions. In this method the RTA model was hand encoded by a team composed of two PDDL experts, using a text editor and relying on dynamic testing. In the ad-hoc method of generate and test, the experts iterate over the following steps: (i) encode requirements, (ii) run a set of planners on several easy problems, (iii) evaluate the resulting plans (if any), and (iv) in the case of strange plans (in relation to the RTA domain requirements) find a way to fix the issue. Usually an expert has to iterate several times through the steps above before plans are produced that match the requirements.

3.2 Method B: itSIMPLE

The main goal of GUI tools is to provide knowledge engineers with a systematic way to reduce modelling time and errors. There are a number of tools available in the research community, such as JABBAH [González-Ferrer *et al.*, 2009] and GIPO [Simpson *et al.*, 2007]. itSIMPLE [Vaquero *et al.*, 2007; 2012] is a method and tools environment that enables knowledge engineers to model a planning domain using the UML standards. The main function of itSIMPLE is to take UML's Object Constraint Language as input through state machine diagrams, and translate them into PDDL.

4 Evaluation of the methods

4.1 Execution

In method A, first the PDDL experts manually coded the RTA domain using PDDL and PDDL2.1 [Fox and Long, 2001], and utilised standard planners such as LPG [Gerevini *et al.*, 2003], and SGPlan [Chen *et al.*, 2006]. The experts translated the informal description of the requirements directly into PDDL, without developing any intermediate notation, using their skill and judgement to perform the encoding. After approximately ten iterations of step (i) and (ii), simple plans were generated which matched requirements. At the next step, longer plans were generated for more complex problems; in this case, unusual behaviour was noticed (e.g., a single ambulance was able to carry 10s of victims) and six more cycles of debugging resulted in plans which solved the test instances.

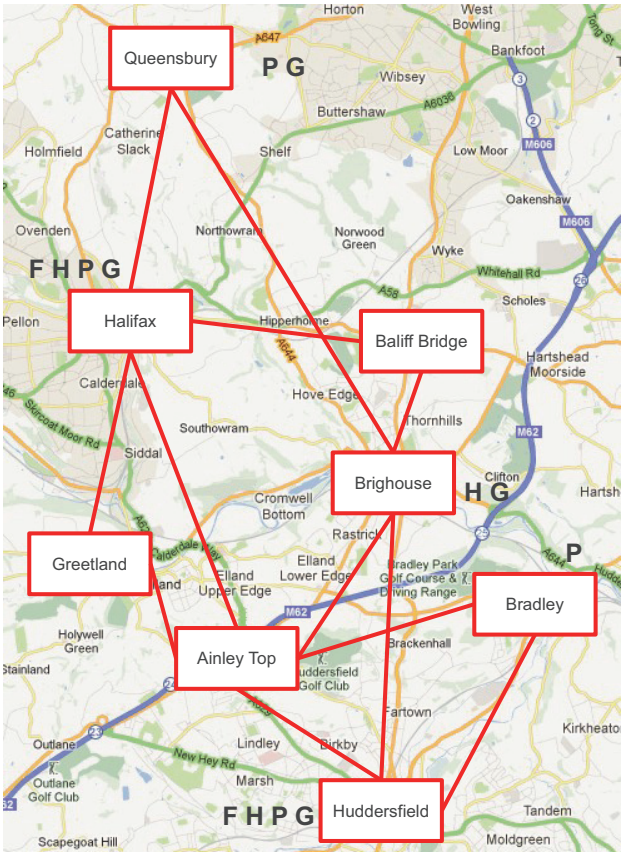

Figure 1: The Road Traffic Domain Model used for empirical analysis. It consists of a portion of the county of West Yorkshire. H, F, P and G respectively stand for Hospital, Fire station, Police station and Garage locations.

The steps in the method B, in general, follow the use of UML in software engineering: (i) design of class diagrams; (ii) definition of state machines; (iii) translation to PDDL; (iv) generation of problem files. A user formulates the requirements by designing several UML diagrams, and automatically generates the corresponding PDDL (or PDDL2.1) domain encoding. While the parameters of the operators and their duration are defined in step (i), in step (ii) the state machine diagrams help the domain modeller to encode pre- and post- condition of operators. In step (iii), we used itSIMPLE to generate both PDDL and PDDL2.1, although it does not cover all the spectrum of timing constraints expressible in PDDL [Vaquero *et al.*, 2012]. The generation of PDDL problem files was done by instantiating objects represented by the previously defined classes, describing their properties in the initial state, and describing the desired properties of objects at the goal state.

The amount of resource to perform the encoding was similar for both the methods. They took approximately 1 person week.

4.2 Process Comparison

Regarding method A, the main issue related with hand coding a real world domain in PDDL is that it tends to be ad-hoc, without the direction or static checks that a tool supported method would impose. The encoding is left to the skill and judgement of the experts that are working on it. This lack of structure leads to domain models that, even if representing the same real world application, are different and hard to understand if developed and maintained by different experts. Moreover, the process of this method is difficult to replicate while everything is left to the sensitivity and to the knowledge of experts. Since the itSIMPLE tool is designed for supporting a disciplined design cycle and for supporting the transition of requirements to formal specifications, the process is more clearly defined and not difficult to repeat.

With respect to bug identification and removal, in the hand-coded method, all but syntactic bugs were dealt with by dynamic testing of the model. Most of the development time was spent in dynamic testing: analysing produced plans, identifying bugs and removing them from the model. While hand encoding a domain, usually many issues are noticed only by carefully reading the generated plans. One example of bug identification is when the team of method A noticed broken vehicles were being delivered to hospitals, instead of garages. Removing the bug in this case amounted to adding further constraints to the operators. On the other hand, most of the time spent with itSIMPLE was in designing classes of objects and defining legal interactions between them. After that, only a relatively short time is required for debugging. However, we found that where debugging was required, it was initiated through dynamic testing: while the structure of the model helps in its development and maintenance, it is the failure of a planning engine to solve a goal which alerts the developer to the presence of a bug, in most cases. This is perhaps a failing peculiar to itSIMPLE, as there are systems with stronger static tests (such as GIPO [Simpson *et al.*, 2007]) which are capable of identifying bugs at an earlier stage than dynamic testing. The need for static tests is reduced, however, given the structure imposed by the UML method; additionally this helps determine the completeness of the model, in terms of classes and finite state machines. Also, itSIMPLE's automated generation of the PDDL model, much like compilation of a high level language into a low level language, has the benefit of eliminating human errors in encoding details. The tool offers the modellers a range of third party planners for generating plans, along with features such as plan analysis, where the plan generated can be viewed graphically.

There are advantages in using hand-coding over using tool supported environments: the development of environments tends to lag behind in the use of expressive modelling languages. itSIMPLE, although being continuously developed, has some limitations in the type of PDDL that it can generate. Also, some details such as parameter associations and metrics are only possible to encode using dialog boxes within GUI-based tools, which hamper their ease of use.

4.3 Product Comparison

For comparing the domain models generated by methods A and B, we selected a subset of the metrics suggested by Roberts and Howe in [Roberts and Howe, 2007]. In this work they described some techniques for predicting the performance of domain-independent planners by evaluating a set of metrics related to both the domain model and the planning problem. Since we are comparing planning domain models for understanding their quality, which depends also on the performance of the planners that will solve the problems, such a set of metrics could give some interesting insights. We considered also the number of lines, which could give a very intuitive idea of the complexity of the models. The results of this comparison are shown in Table 1, metrics considered are the number of types, predicates and operators, the mean number of parameters per operator, the mean number of pos/neg preconditions and the mean number of pos/neg effects.

Metric	PDDL		PDDL2.1	
	A	B	A	B
# types	19	22	19	22
# predicates	16	18	16	16
# operators	12	14	12	12
mean parameters	3.1	3.2	3.1	3.2
mean precond+	4.3	3.8	4.3	4.0
mean precond-	0.2	0.4	0.2	0.4
mean eff+	1.7	1.4	1.7	1.6
mean eff-	1.9	1.2	1.9	1.3
# lines	225	263	248	259

Table 1: The values of the metrics selected for comparing the domain models generated by methods A and B.

We found that the iterative process in Method A led to an over-constrained domain encoding. Many of the constraints, added in the form of pre- and/or post- conditions, were built up incrementally during de-bugging in an ad-hoc fashion, in order to avoid unwanted behaviours. The resulting model is complex and hard to read and understand compared to the model developed using method B. That method A leads to a constrained model is confirmed by the higher mean number of positive preconditions and effects. This is not noticeable by the number of lines of the files because method B invites to use many different types, as usual in KE approaches, which are not listed in a very compact way. The PDDL domain model generated by method B has 2 more actions than the method A one; these operators are related to the untrapping people and extinguish fire tasks and are used for avoiding that the same fire brigade extinguishes several fires or untraps several people at the same time. The method A model exploits a "trick": the PDDL experts added the same predicate as a positive and negative effect of the operator, which avoids the simultaneous execution of actions instantiated with similar parameters. Although these kind of tricks are commonly used by experts, their impact on the performance of the planners have not been studied, and moreover they make the domain model harder to read and understand.

We observed that the structured and principled process of encoding the requirements in Method B led to domain encodings that are clear and easy to understand. Moreover, we found that the UML documentation is useful in maintenance, as it helps trace the encoding to the initial requirements. The main difference between PDDL and PDDL2.1 encodings is

LPG						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.03	0.03	97	90	28	29
30P, 10V, 2T, 1F	0.5	0.3	318	317	90	108
100P, 30V, 5T, 3F	35.6	22.8	1015	1001	350	311
100P, 30V, 5T, 3F *	62.4	37.9	1033	988	254	246
SGPLAN						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.12	0.09	95	96	95	96
30P, 10V, 2T, 1F	0.36	0.59	324	338	324	338
100P, 30V, 5T, 3F	1.25	1.50	998	1018	998	1018
100P, 30V, 5T, 3F *	2.85	3.60	993	1068	993	1068

LPG						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.04	0.03	94	91	82	76
30P, 10V, 2T, 1F	0.7	0.3	317	321	198	220
100P, 30V, 5T, 3F	57.8	25.2	1000	1012	530	526
100P, 30V, 5T, 3F *	86.0	42.1	1025	1029	315	330
SGPLAN						
Instance	**CPU time**		**# Actions**		**Duration**	
	A	**B**	**A**	**B**	**A**	**B**
6P, 5V, 1T	0.11	0.13	88	97	117	141
30P, 10V, 2T, 1F	0.48	0.54	331	332	528	415
100P, 30V, 5T, 3F	1.45	1.81	1006	1048	1634	1115
100P, 30V, 5T, 3F *	3.64	4.40	1000	1062	1543	1322

Table 2: For every instance, the CPU time (seconds), the number of actions and the duration of plans generated by LPG and SGPlan on domains encoded using methods A and B. The upper table refers to PDDL encodings, the lower to PDDL2.1. Instances are described by the number of victims (P), the number of vehicles involved (V), the number of victims trapped (T) and the number of cars on fire (F); * indicates that the number of available emergency vehicles is doubled.

that PDDL plans, since actions are instantly completed, are a very compact version of the PDDL2.1 ones. The simpler encoding is not very realistic, however, as emergency vehicles are used without taking into account their distance from the accident location, since distance cannot be described in the simpler encoding.

To compare the operationality of the products, we investigated the performance achieved by planning systems on the models generated exploiting by methods A and B. We ran LPG and SGPlan on a set of test instances using the different models. We selected them due to their ability to handle durative actions and negative preconditions, which are both used in the generated domains, because they are readily available and performed well at IPCs. The results of the experiment are shown in Table 2 in terms of CPU time, number of actions and plans duration.

These results indicate that LPG with the hand written domain models needs more CPU time, both in PDDL and PDDL2.1, than with the models generated through method B. In the number of actions and duration of the plans there are no significant differences. The performance of LPG while exploiting method B models is very interesting; for generating good quality solutions, it requires significantly less CPU time.

On the other hand, SGPlan displays a very different performance profile compared to LPG. In this case, the domains encoded by method B slow down the plan generation process, but method B encodings lead to plans with significantly shorter makespan when generated by LPG. While SGPlan is faster than LPG at plan generation, it was not effective at exploiting the parallelisation of actions in solution plans, which unlike in LPG's performance, resulted in plans with a high makespan.

5 Conclusions and Future Work

In this paper we have developed requirements for a new planning domain, the RTA domain, addressing the problem of managing emergency situations in road traffic accidents. We have elicited a set of requirements, and used domain analysis to make precise and unambiguous relevant features for the planning problem. We then described two methods used for formulating requirements into domain models, and set up an evaluation experiment where they were used to design and create RTA domain models. Special attention was given to knowledge engineering aspects such as how long it takes to create a model or which tools can be used to verify the model. We observed that creating different models does not take very different amounts of time (taking into account the developers' expertise). We also noticed that most of the existing domain-independent planners do not support many features required for modelling real world situations: i.e., negative preconditions and durative actions. This is, clearly, a big limitation for their application. The main outcome from our work to feed back to tools developers is to provide facilities to couple planning engines and formulation tools (see [Shah *et al.*, 2013] for more details of such lessons learned).

As for the comparison, we can conclude that using itSIMPLE (method B) achieved superior results in both process and product metrics. From the process point of view, method B is easier to replicate and does not require high expertise in planning languages. From the product point of view, models are clearer to read, understand, and easier to maintain using method B. Moreover, the domain model produced with it led to a better performance from both the selected planners, even if on different metrics: LPG is significantly faster in plan generation, and SGPlan generates better quality plans, using method B's domain model.

Given the fact that different planners exploit different search techniques, they could have very different performance on the same domain encoding, as shown in our experimental analysis. The strategy that we suggest, that is derived from the experience gathered in this work, is (i) to define a metric to be optimized, (ii) select a (set of) planner(s) which handle the required features, (iii) test the planners on some easy instances, and (iv) selecting the planners, or the set of planners, which achieves the best results w.r.t. the predefined metric.

Future work will involve a simulation framework for eval-

uating plan execution, where we can couple model design and plan generation more tightly. This may reveal opportunities for improving domain models in general, and the RTA model in particular. We are also interested in simulating more complex road accidents, with blocked roads or accidents occurring in locations difficult to reach (e.g. on narrow roads). Moreover, we should consider more expressive approaches, for instance, PDDL+ [Howey *et al.*, 2004], capturing features of continuous planning since it might produce more robust system working in real-time and be able to react to unexpected events. Another interesting area might be to compare our centralised approach to using a multi-agent approach which moves the problem from centralized to a distributed point of view.

References

[Barreiro *et al.*, 2012] J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkayloz, P. Morrisy, J. Ong, E. Remolina, T. Smith, and D. Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12) – The 4th International Competition on Knowledge Engineering for Planning and Scheduling*, 2012.

[Barták *et al.*, 2010] R. Barták, S. Fratini, and T.L. McCluskey. The third competition on knowledge engineering for planning and scheduling. *AI Magazine*, 31:95 – 98, 2010.

[Benesch, 2011] Benesch. *Traffic Incident Management Operations Guidelines*. Iowa Department of Transportation, Faderal Highway Administration, 1200 New jersey Avenue, SE, Washington, DC 20590, March 24 2011.

[Chen *et al.*, 2006] Y. Chen, B.W. Wah, and C. Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)*, 26:323–369, 2006.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fox and Long, 2001] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains . In *Technical Report, Dept of Computer Science, University of Durham*, 2001.

[Gerevini *et al.*, 2003] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)*, 20:239 – 290, 2003.

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

[González-Ferrer *et al.*, 2009] A. González-Ferrer, J. Fernández-Olivares, and L. Castillo. JABBAH: A java application framework for the translation between business process models and htn. In *Working notes of the 19th International Conference on Automated Planning & Scheduling (ICAPS-09) – Proceedings of the 3rd International Competition on Knowledge Engireeng for Planning and Scheduling(ICKEPS)*, pages 28–37, 2009.

[HA, 2009] HA. *Highways Agency Network Management Manual*. Highways Agency, Network Services, Network Management Policy Team, City Tower, Piccadilly Plaza, MANCHESTER M1 4BE, 5.10 edition, July 2009.

[Howey *et al.*, 2004] R. Howey, D. Long, and M. Fox. Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings of the Sixteenth International Conference on Tools with Artificial Intelligence*, pages 294 – 301, 2004.

[McDermott, 1998] J. McDermott. 1998 AIPS planning competiton. ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html, 1998.

[Owens *et al.*, 2000] N. Owens, A. Armstrong, P. Sullivan, C. Mitchell, D. Newton, R. Brewster, and T. Trego. Traffic Incident Management Handbook. Technical Report Office of Travel Management, Federal Highway Administration, 2000.

[Roberts and Howe, 2007] M. Roberts and A. Howe. Learned models of performance for many planners. In *Proceedings of the ICAPS-07 Workshop of AI Planning and Learning (PAL)*, 2007.

[Shah *et al.*, 2013] M.M. Shah, L. Chrpa, F. Jimoh, D. Kitchin, T.L. McCluskey, S. Parkinson, and M. Vallati. Knowledge engineering tools in planning: State-of-the-art and future challenges. In *Submitted to the Knowledge Engineering for Planning and Scheduling workshop – The 23rd International Conference on Automated Planning & Scheduling (ICAPS-13)*, 2013.

[Simpson *et al.*, 2007] R.M. Simpson, Diane E. Kitchin, and T.L. McCluskey. Planning domain definition using gipo. *Knowledge Engineering Review*, 22(2):117–134, June 2007.

[Vaquero *et al.*, 2007] T.S. Vaquero, V. Romero, F. Tonidandel, and J.R. Silva. itSIMPLE2.0: An integrated tool for designing planning domains. In *Proceedings of the 17th International Conference on Automated Planning & Scheduling (ICAPS-07)*, pages 336–343. AAAI Press, 2007.

[Vaquero *et al.*, 2011] T.S. Vaquero, J.R. Silva, and J.C. Beck. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop – The 21th International Conference on Automated Planning & Scheduling (ICAPS-11)*, 2011.

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