Knowledge Engineering Techniques for Automated Planning

Mohammad Munshi Shahin Shah

Department of Informatics

University of Huddersfield

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

March 2014

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. Also, there is little published research to inform engineers on which method and tools to use in order to effectively engineer a new 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. 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 an ad hoc process, 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.

The major contribution of this thesis is the evaluation of knowledge engineering tools and techniques involved in the formulation of knowledge. To support this, we introduce and encode a new planning domain called Road Traffic Accidents (RTA), and discuss a set of requirements that we have derived, in consultation with stakeholders and analysis of accident management manuals, for the planning part of the management task. We then use and evaluate three separate strategies for knowledge formulation, encoding domain models from a textual, structural description of requirements using (i) the traditional method of a PDDL expert and text editor (ii) a leading planning GUI with built in UML modelling tools (iii) an object-based notation inspired by formal methods. We evaluate these three approaches using process and product metrics. 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.

In addition, we discuss a range of state-of-the-art modelling tools to find the types of features that the knowledge engineering tools possess. These features have also been used for evaluating the methods used. We benchmark our evaluation approach by comparing it with the method used in the previous International Competition for Knowledge Engineering for Planning & Scheduling (ICKEPS). We conclude by providing a set of guidelines for building future knowledge engineering tools.

Declaration

(i) The author of this thesis (including any appendices and/or schedules to this thesis) owns any copyright in it (the ”Copyright”) and s/he has given The University of Huddersfield the right to use such copyright for any administrative, promotional, educational and/or teaching purposes.

(ii) Copies of this thesis, either in full or in extracts, may be made only in accordance with the regulations of the University Library. Details of these regulations may be obtained from the Librarian. This page must form part of any such copies made.

(iii) The ownership of any patents, designs, trademarks and any and all other intellectual property rights except for the Copyright (the ”Intellectual Property Rights”) and any reproductions of copyright works, for example graphs and tables (”Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property Rights and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property Rights and/or Reproductions

To my loving parents Billal Hossain and Rezia Khaton.

*It is because of you that I am **ME**!*

Acknowledgements

Above all, I praise almighty Allah for his generosity, compassion and mercifulness, for giving me all the opportunities and abilities to finalize this PhD thesis.

I am truly grateful to my supervisor Lee McCluskey, whose helpful guidance, insightful vision, and continuing support have lead me to grow greatly over the years of my PhD. Lee's ability to identify motivating problems, his principles, and hard work are contagious, allowing his students to achieve their full potential. I am very happy to be one of these students, and I am thankful for the opportunity to learn from him. Thank you Lee!

One of the most interesting and informative elements of my PhD career has been the interaction with the members of PARK (former KEII) research group. I have extremely enjoyed being part of one of the most collaborative and productive groups. I am proud to get the opportunity to share this experience with Salihin Shoeeb, Ali Fanan, Aisha Jilani, Munir Naveed, Rabia Jilani, Falilath Jimoh, Asma Kilani, Margaret West, Diane Kitchin, Ilias Tachmazidis, Simon Parkinson, Maro Vallati, Lukas Chrpa, Grigoris Antoniou and Lee McCluskey.

I am thankful to every one of my co-authors Peter Gregory, Lukas Chrpa, Mauro Vallati, Falilat Jimoh, Simon Parkinson, Margaret West, Diane Kitchin and Lee McCluskey over the last years, from whom I learned about new research directions, open problems, solution techniques, high standards, writing style, and the fun of pursuing challenging and ambitious research goals. Their efforts in researching new directions and applications, and creating an efficient implementation of our methods have fundamentally improved the work in this thesis. I also like to thank the reviewers of

my research papers for their insightful comments and suggestion, which results distinctive outcome.

This research was carried out with the scholarship provided by the University of Huddersfield. I would like express my appreciation to the University and all the staff related to this project directly or indirectly. I like to thank the University authority for giving me this scholarship, opportunity to take professional development courses and approval for conference travel grants.

I would not be at Huddersfield if Alamgir Hossain, who believed in me and invited me to the MSc by Research at the University of Bradford in 2005. I am grateful to Alamgir for this opportunity, which had such and a greater positive impact on my life. I am also grateful to my friend Pranab Tusher for pushing me to go to Bradford and being there for me all the time. It would have been difficult to pursue my PhD without the moral support of my friends specially Mehedi Siddiqui, Abdul Baten, Farhana Moshira, Samsun Nahar, Shanu Chaudhury and Ruhul Amin over the years of my education career. I am really thankful to you all. I am also thankful to all my friends, who I have forgotten to mention here. My best wishes for them.

I would like to thank my family for their endless love, support and encouragement. I am particularly grateful to my parents (Billal Hossain and Rezia Khatoon), sister (Mahamuda Parveen), brother Mizanur Rahman, and brother-in-law (Mosharraf Hossain). I will never forget your care and love that have been immutable over my life. Thank you all from the bottom of my heart!

Finally, I am deeply thankful to my wife Shahida Sams for her love, care, and patience over the years of my PhD. Thank you both for being there for me all the time. Also, my appreciation goes to my son Shayan Shah for allowing Abbi to spend time studying when Shayan wanted me most. I love you for this.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Automated Planning research is one of the major research disciplines in the area of Artificial Intelligence (AI) for more than thirty years. It has a wide range of applications in areas such as rover mission to Mars (Bresina et al., 2005), business processes modelling (González-Ferrer et al., 2009), logistic support (Tate et al., 1996), multiple battery usages (Fox et al., 2011), machine tool calibration (Parkinson et al., 2011) and computer games (Naveed et al., 2012). These applications are rapidly growing in size and complexity beyond the capabilities of the people responsible for planning. Also, there are challenges in the automation of real-world application: such as reducing the cost of planning techniques, and improving the quality of resulting plans and preferences.

Traditionally, planning systems have fallen into two major classes: domain independent and domain-dependent. Planning system designed for solving the problem of a specific application is called a domain-dependent. Reliable planning systems are

very important in the real-world applications. AI planning research shares many common aspects although diverse in nature. Therefore, it is possible to build general approach for planning systems to process and specify the application domain model and produce solution for any given problem of that domain (Erol, 1995). Building domain-independent planning systems is very challenging (Ghallab et al., 2004).

The popular planning system is stemmed from STRIPS (Fikes and Nilsson, 1971a) planning system, which uses primitive action descriptions to encode and solve propositional representations of planning problems. The Graphplan system was developed by Blum and Furst (Blum and Furst, 1997a) and SATPlan developed by Kautz and Selman (Kautz, 2006; Kautz and Selman, 1996, 1999) had greatly improved the efficiency of this form of planners. However, these planners do not incorporate domain specific control knowledge, but instead rely on efficient graph-based or propositional representations and advanced search techniques, whilst they could only solve simple, small toy problems from the standard testing STRIPS-style domains used for AIPS planning competition (Bacchus, 2001).

Real-world planning problems require expressive knowledge representations than the traditional benchmark problems. To apply planning techniques to the real-world, it is necessary develop techniques with the capability of capturing rich knowledge model (Wilkins and desJardins, 2000). Domain knowledge capturing and modelling has been an issue in planning research as early as the work on the NONLIN (Tate, 1977) planner, which was later developed as a Hierarchical Task Network(HTN) Planning. HTN planning enables multiple layers of abstraction in the domain definition. Generally, an HTN planner attempts the top level abstract layer of the problem and decompose until the lower level (often called primitive operators). It differs from the normal precondition planner in that it eventually splits disjunction into the search space with the

help of pre-reduction refinements, considering each way of reducing the non-primitive task in a different search branch (Kambhampati, 1997). Encoding knowledge for HTN reduces the search space significantly.

To solve practical planning problems HTN has been proved as the most successful in compare to other planning techniques (Drummond, 1994; Erol et al., 1994; Nau, 2007). Generally, various types of domain knowledge can be encoded in HTN techniques. For example, SIPE-2 (Wilkins, 1999), one of the best-known knowledge intensive applications of HTN that includes multiple levels of abstraction in the knowledge representation according to user requirements (Wilkes and Myers, 1995) which assists the planner to control the search space during the plan generation and execution.

On the other hand, the standard domain modelling language such as PDDL (Planning Domain Definition Language) (Ghallab et al., 1998; McDermott, 1998) is an action-based language. It is an expressive language that allows temporal constraints and numeric fluents in the version PDDL2.1 (Fox and Long, 2001). Advantages of using PDDL2.1 (Fox and Long, 2001) is that it accepts numerical constraints to perform arithmetic operations, such as evaluating distances and time required for movements, and use *durative actions* in order to facilitate temporal constraints in planning. The PDDL domain modeling language has been extended to level five for capturing domain knowledge from continuous domain, so called PDDL+ (Howey et al., 2004).

Formulating domain model for real-world application is a laborious process that requires the knowledge engineer to be the domain expert as well as planning language. The current trend is to use the hand-encoding technique for knowledge engineering but there is strong support and reasons to use tools to embed software lifecycle (Chien et al., 1996). A compositional and model-based domain modelling language has been introduced by NASA which supports the concurrent transition systems

models (Muscettola et al., 1998) in order to widen the applicability of AI techniques to real-world applications. This kind of formalism usually builds individual model components and uses a set of mode for transition from one model to another. The goal of this approach to allow the non-AI expert to inspect the model and understand the knowledge in the system as this approach draws a clear line between the models and heuristics.

The idea of Knowledge engineering (KE)[1] for automated planning is to formulate, acquire, validate and maintain the domain knowledge, where a key product is the *domain model* containing the declarative description of domain dynamics (Biundo et al., 2002). The field has advanced steadily in recent years, helped by a series of international competitions, the experience from planning applications, along with well developed support environments (for example, *Europa* (Barreiro et al., 2012), itSIMPLE (Vaquero et al., 2007, 2010, 2009, 2012), GIPO (McCluskey et al., 2003; Simpson et al., 2007, 2001)). 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., 2011b). While these surveys 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. On the other hand, for a new planning domain (not been modelled for planning application), 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

[1] KE will be used throughout the thesis

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, i.e., the encoder must know the planning language in expert level and must be an AI planning expert. This process likely to be inefficient since it has to rely on dynamic testing for debugging. On the other hand, the use of tools such as itSIMPLE (Vaquero et al., 2012) or GIPO (Simpson et al., 2007), 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. itSIMPLE integrates post design framework (Vaquero et al., 2011a,c) to identify hidden knowledge that directly impacts on the planning systems. Post-Design framework also aims to understand the plan quality (good or bad) and the reason of such quality (Vaquero et al., 2011a).

Although, Knowledge Engineering (KE) issues: evaluation of knowledge engineering methods, improved representation languages, standard language and ontology, have been identified by McCluskey (McCluskey, 2002) but there is a little or no work on Knowledge Engineering for Artificial Intelligence Planning & Scheduling (KEPS) tools and techniques have been done to identify the best way to design and develop a *knowledge model* (domain model). This thesis is motivated by the need for the Knowledge Engineering (KE) tools and techniques for automated planning application to cover the issues and challenges that are not currently addressed. There is also a need to understand the capabilities of current KE tools and techniques to design and develop a real-world domain model. In this work, we focus on two important research challenges encountered in KE: i) analyse the process of encoding domain models using KE tools and hand-encoding methods in order to find the efficient method and ii) the

performance of the product (domain model) that have been developed using different methods during the plan generation.

1.2 Contribution of the thesis

This work contributes to the area of research into knowledge engineering for real-world planning domains in the following ways.

- an evaluation of the relative merits of hand coding planning domain models vs those produced by tools.
- an overview of existing KE tools and techniques have been provided to evaluate the types of features used in modelling tools, the support that they provide, efficiency of using, and the kind of bugs they discover.
- a set of requirements and insights for future tools which can be used to create PDDL domain models,
- the planning requirements, and domain model, of a new real-world planning domain, the Road Traffic Accident (RTA) domain, used as a Case Study to inform the evaluations

1.3 Structure of the Thesis

This thesis is structured in seven chapters including the current one. The current chapter (Chapter 1) briefly introduces the problem research objectives. The rest of this thesis is organised as below -

Chapter 2: This chapter provides background of knowledge engineering in the context of automated planning and scheduling. We first discuss the history of planning and the associated domain modelling languages. We also emphasis on the knowledge engineering tools and techniques and how these could be helpful for overcoming the knowledge engineering problems.

Chapter 3: This chapter reviews and analysis the existing knowledge engineering tools and techniques that support designing of AI planning applications. This gives deep insights of the design process of the knowledge model and how the introduction of Knowledge Engineering techniques could address the problems related to the knowledge modelling of planning application. This discussion also finds a number features of Knowledge Engineering tools and techniques which may play significant role in designing future tools.

Chapter 4: This chapter introduces a new real-world planning domain, the Road Traffic Accident(RTA). After a careful investigation, a set of requirements has been derived, and using domain analysis to make precise and unambiguous relevant features of the planning problem.

Chapter 5: In this chapter we present three different models of Road Traffic Accident (RTA) domain derived from the requirements that have been described in Chapter 4.

The first model is developed using traditional method of modelling PDDL (McDermott, 1997) domain using a text editor which relies on dynamic testing for debugging.

The second model is developed using itSIMPLE, a state-of-the-art PDDL domain modelling tool that uses GUI with built-in UML (Booch et al., 1997) modelling leads a domain modeller through the formulation of requirements by designing several UML diagrams, and automatically generates the corresponding PDDL.

The third model is developed using GIPO, a rigorous method utilising a hierarchical, object-based notation OCL_h. This tool gives the modeller the facility to encapsulate the syntax of the language by some graphical interface.

Chapter 6: This chapter evaluates the knowledge engineering techniques by using the domain construction experience form three distinct methods.

We discuss some criteria to evaluate the knowledge engineering tools and techniques to derive a guideline of future tool.

Chapter 7: This chapter summarises the contribution of this thesis. also we discuss the limitation of this research and provide future work in the knowledge engineering for automated planning.

1.4 Publications

This section presents the list of publications that were achieved during my PhD research in order to support and validate the scientific quality of this thesis.

1. M. M. S. Shah, D. Kitchen, T. L. McCluskey, L. Chrpa and M. Vallati, (2013), *Exploring Knowledge Engineering Strategies in Designing and Modelling a Road Traffic Accident Management Domain*, In the Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-13), August 3-9. Beijing, China.

2. M. M. S. Shah, L. Chrpa, F. Jimoh, D. Kitchin, T. L. McCluskey, S. Parkinson and M. Vallati (2013), *Knowledge Engineering Tools in Planning: State-of-the-art and Future Challenges*, In the Proceedings of ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS), June 10-14, Rome, Italy.

3. F. O. Jimoh, L. Chrpa, T. L. McCluskey and M. M. S. Shah (2013), *Towards Application of Automated Planning in Urban Traffic Control*, In the Proceedings of 16^{th} International IEEE Annual Conference on Intelligent Transportation Systems, October 6-9, The Hague, The Netherlands.

4. M. M. S. Shah, TL McCluskey, L. Chrpa (2012), *Symbolic Representation of Road Traffic Domain for Automated Planning to Manage Incidents*. In the proceedings of 30th Workshop of the UK Planning And Scheduling Special Interest Group (PlanSig12), University of Teesside, Middlesbrough, UK.

5. M. M. S. Shah, T. McCluskey, P. Gregory and F. Jimoh (2012), *Modelling Road Traffic Incident Management Problem for Automated Planning*, 13^{th} IFAC Symposium on Control in Transportation System (CTS12), September, 12-14. Sofia, Bulgaria.

6. M. M. S. Shah, L. Chrpa, P. Gregory, T. L. McCluskey and F. Jimoh (2012), *OCLplus: Processes and Events in Object-Centred Planning*, the 6^{th} Starting Artificial Intelligence Research Symposium, August, 27-28. Montpellier, France.

7. M. M. S. Shah, T. L. McCluskey and M. M. West,(2009), *A Study of Synthesizing Artificial Intelligence (AI) Domain Models by using Object Constraints*. In the Proceeding of the Computing and Engineering Annual Researchers Conference (CEARC '09), December, University of Huddersfield, UK, ISBN: 978-1-86218-085-7.

8. M. M. S. Shah, T. L. McCluskey and M. M. West, (2009), *An investigation into using Object Constraints to Synthesize Planning Domain Model*, In Doctoral Consortium of 19^{th} International Conference on Automated Planning and Scheduling (Refereed paper), September, Thessaloniki, Greece.

Chapter 2

Background

In this chapter we present a historical overview of Knowledge Engineering (KE) for AI planning literature in order to provide a context within which the contributions of this thesis can be evaluated. It is not intended as a comprehensive review of the AI planning, but to provide a closer look on the AI planning application.

We first discuss the development history of automated planning, and a brief overview of the planning techniques and languages. The aim of this chapter is to discuss the Knowledge Engineering (KE) for automated planning. We overview general understanding of KE for planning including the KE process and language. The core of this chapter examines the design process, issues and advanced in the KE research. The discussion mainly focuses on the knowledge engineering techniques for planning, where the work described in this thesis fits in, and how it relates to the recent and current research efforts.

2.1 History of Automated Planning

The beginnings of AI planning can be traced back to the GPS system, a General Problem Solver system developed by Allen Newell, J.C.Shaw and Herbert A.Simon (Newell and Simon, 1963). It developed a ”means-end” search strategy, solved a problem by setting up (Pearson and Laird, 1996) subgoals which reduced the differences between the current state and the goal state, and achieving those in order to achieve the original goal. McCarthy and Hayes developed the *situation calculus* (McCarthy and Hayes, 1969) using first-order predicate logic to reason about the actions. The advantage of the situation calculus is that it provides a theoretical framework to represent actions on a clear semantics.

An important landmark in the development of the planning system was the STRIPS system (the Stanford Research Institute Problem Solver), which was developed by Fikes and Nilsson in the late nineteen sixties (Fikes and Nilsson, 1971a) to control the movement of a robot called Shakey, which pushes various boxes through several interconnecting rooms. In the STRIPS style planning, actions are represented as preconditions and effects, as add-list and a delete-list. The STRIPS representation avoids the frame problem by making closed world assumption of default persistence to simplify the planning process. The STRIPS representation formalism is still used today and has influenced the design of more expressive planning languages.

The efficiency of the search process can be significantly affected by the order in which simultaneous goals are attempted. Sacerdoti (Sacerdoti, 1974) introduced a hierarchical abstraction planning system ABSTRIPS on the base of STRIPS. A Hierarchical planner works at different levels of abstraction. It solves a higher, abstracted problem first, then plans at increasing levels of detail, moves from the abstract to the

concrete. NOAH (Sacerdoti, 1975) was the first hierarchical planner that allowed to develop plans to represent the partial order of actions considering time. Before this, the operators in the partially - constructed plan were always given a total ordering. The advantage of a partial order representation is that a planner may be able to consider fewer plans, and thus be able to generate a solution to a problem more quickly.

A notable successor of NOAH was Tate's NONLIN (Tate, 1977) which used the so called Hierarchical Task Network(HTN) planning technique. HTN performs search for plans that accomplish task networks (Erol, 1995; Kambhampati, 1994). Current HTN planners are different from previous versions as they allow the user to describe and limit the type of the search space.

Later, the Graphplan system developed by Blum and Furst (Blum and Furst, 1997a) and SAT-plan developed by Kautz and Selman (Kautz and Selman, 1996) provided insight into improving planning efficiency. However, these planners rely on efficient graph-based or propositional representations and advanced search techniques instead of incorporating domain specific control knowledge. They could only solve simple and small toy problems from the standard testing STRIPS-style domains used for the IPC planning competition(Bacchus, 2001).

International Conference on Planning and Scheduling (ICAPS)[1] has begun to organise International Planning Competition (IPC)[2] since 1998 in order to improve the track record of automated planning algorithms. The IPC is a biannual competition where planners challenge each other to win based on a number of given criteria. FF (Hoffmann and Nebel, 2001), the fast-forward planner, is one of the most successful planning algorithms won the 2^{nd} IPC. FF is a state space planner using forward

[1] http://www.icaps-conference.org/
[2] http://ipc.icaps-conference.org/

chaining heuristics to estimate the goal. FF is considered as an advanced successor of HSP system (Bonet et al., 1997) with significant differences. FF was extended to Metric-FF (Hoffmann et al., 2003) to deal with numeric constraints. In the same competition SHOP2 (Nau et al., 2003), a hierarchical domain independent planner won top four awards for outstanding performance. Using local search and planning graph, LPG (Gerevini et al., 2003) won the the competition on 3^{rd} and 4^{th} IPCs which also can handle the numeric constraints and durative actions. In the 5^{th} IPC, the sat-based planner SATPlan (Kautz, 2006) and MAXPLAN (Xing et al., 2006) won award for the best performance in optimal planning track. In the same competition the SGPlan (Chen et al., 2006) which decomposes the goal into subgoals to solve using FF-like techniques. In the IPC-6 LAMA (Richter and Westphal, 2008), which is based on FF-based heuristic, outperformed in the sub-optimal track. In the IPC-7, LAMA-11 (Richter et al., 2011) won the award on the satisfying track but could perform well in the multi-core track where ARVANDHERD (Nakhost et al., 2011) won the award.

Automated planning research has been moving away from its restricted classical domain by directing many new techniques for handling real-world application. The last decade of Planning focuses on many new areas like time and resources, numeric computations involving resources, probabilities, geometric and spatial relationships, which opened new research areas to deal with real application (Nau, 2007). There was a special track of continuous planning in the ICAPS12 Conference. Continuous planning emphasises a number of different planning tasks online, planning with continuous change, management of continuous resources, real-time behavior and mixed discrete-continuous dynamics. This is intended as a multi-disciplinary track that focuses on all elements of online systems that perform real-time planning, execution, monitoring

and adaptation. In the ICAPS13[1] there is a special track for 'Novel Applications' to promote scientific research in applied planning.

2.2 Planning Techniques

2.2.1 STRIPS Planning

STRIPS (Fikes and Nilsson, 1971a), the Precondition achievement planning system is the most general classical planning technology. The precondition-effect representation places no assumptions upon the structure of the domains to which it is applicable, leaving the complexity of planning to the domain-independent planning algorithm. STRIPS-Style planning is considered to be straightforward search problem for an agent which identifies all possible orders for an action can possibly perform depending on the given constraints for particular world. The main features of STRIPS planning systems are -

- Supports Planning Domain Definition Language (PDDL) family languages
- Problem solves by search
- Forward State-Space search
- Backward State-Space search
- Supports STRIPS planners

In the following two subsections we discuss two STRIPS style planners called LPG and SGPlan in order to use them for evaluation. LPG and SGPlan planners were used

[1] http://icaps13.icaps-conference.org/

because of their ability of handling durative actions and negative precondition. Also, they are readily available and performed well at International Planning Competition (IPC). There are a number of better performed domain independent planners, such as FF (Hoffmann and Nebel, 2001), FF-Metric (Hoffmann et al., 2003) or Fast Downward (Helmert, 2006), available in the planning community for evaluating RTA domain. In our evaluation we need planners which can handle both classical and temporal domain models. FF and Fast Downward Planners can only handle classical PDDL domain model. On the other hand, FF-Metric is capable of handling both classical and temporal PDDL domain model, which is also the winner of 2^{nd} IPC. Both LPG and SGPlan execute FF-style search and use more advanced techniques for finding plan. For example, LPG is capable of switch best-first search and restarts automatically, and SGPlan uses subgoals for bigger problems. LPG and SGPlan won 3^{rd} and 5^{th} IPC respectively.

2.2.1.1 LPG

Local search for Planning Graph (LPG) is a domain independent planner that based on local search and planning graph. LPG can handle PDDL2.1, which involves durative actions and numerical values. It is a multipurpose planning system that allows generating and repairing plans, and incremental planning (Fox et al., 2006) in PDDL2.2 domains (Hoffman and Edelkamp, 2005). This planner uses stochastic local search procedure to explore a search space of partial plans represented as *linear action graphs*, the variants of famous planning graph (Blum and Furst, 1997b). LPG is a domain-independent planner that has the advantages of estimating heuristic by exploiting planning graph. This planning system can produce quality plans using one or more criteria that can be accomplished by an anytime process to produce a sequence of action. This system comes with a FF style best-first algorithm. LPG is capable of switching to

best-first search after a certain number of search steps and restarts automatically.

2.2.1.2 SGPlan

SGPlan (Chen et al., 2006; Hsu and Wah, 2008) is a 'Subgoal Partition' planner to solve larger problems efficiently. The main goal of SGPlan is to reduce the search space of the original problem significantly. This planner fully supports many versions of PDDL, such as, PDDL2.1, PDDL2.2 and PDDL3.0. There are advantages of using subgoal partition as it involves a significantly smaller number of constraints than the original problem. Also, planning landmarks analysis is used to decompose the hierarchy of subproblems. Planning landmarks are facts that must be true at some point in every solution plan, which are effective for solving larger problems. In the implementation, SGPlan uses modified Metric-FF planning strategy for basic planning.

2.2.2 HTN Planning

Actions and states definition in HTN planning are similar to the STRIPS representations. The main difference between HTN planners and Precondition Achievement Planning is in what they plan for, and how they plan for it. HTN planners generate plans by searching over the task networks which is a collection of tasks. All the collection of tasks could be primitive in the sense that they cause a simple state transition to the world, or they may contain compound tasks that could be decomposed during the planning of primitive tasks (Ghallab et al., 2004). Essentially, an HTN may represent a plan ranging from a partial plan to a fully instantiated plan.

In HTN, a task can be a primitive task, a compound task, or a goal. A primitive task corresponds to some operations that can be readily done and can change the state of

the world. A compound task is decomposable to another compound task or primitive tasks. A goal is something that needs to be achieved by assigning a compound task or a primitive task. The decomposition of compound tasks is accomplished by a method (Tate, 1977). A method is a construct associating a compound task to a task network consisting of sub-tasks of the compound task. Conceivably, one can have multiple methods for each compound task. Finally, an operator is a construct that consists of a primitive task, pre-conditions, and post-conditions (Tate et al., 1994).

In this following subsection we discuss *HyHTN*, a hybrid planner which is also the planner integrated with GIPO.

2.2.2.1 HyHTN

HTN planning process starts from high level actions then decomposes them into sub-tasks at different levels of abstraction. *HyHTN* (McCluskey et al., 2003) expands the task networks with the combination of HTN planner and preconditions achievement planning. It starts planning from the initial world states, expands the problem task networks by the orders given by user or domain specification, and then further expands the action until it is executable. *HyHTN* is a planner which has achieved a high level of performance. The representation language of *HyHTN* planner is *OCL* which is an expressive language that allows to build an HTN and/or a pre-condition planner input (McCluskey et al., 2003). The *HyHTN* as hybrid planner that has two significant features:

- the main search procedure of *HyHTN* are fast-forward for precondition planning and an HTN reduction for hierarchical reduction
- the domain model fed to the *HyHTN* planner can be encoded using a Knowledge

Engineering tool called GIPO (McCluskey et al., 2003; Simpson et al., 2007).

2.3 Knowledge Engineering

Knowledge Engineering for Planning & Scheduling (KEPS) is concerned with the design process of a domain model using a Domain Definition Language by using appropriate methods for knowledge model development and their respective integration with the Artificial Intelligence Planning algorithms. KEPS was defined by McCluskey in the 2003 PLANET Roadmap (Biundo et al., 2002; McCluskey, 2000a) as the processes to acquire, validate and verify, and maintain of planning domain models by the selection and optimisation of appropriate planning machinery to make up a planning and scheduling application. Although KEPS is considered as a special case of Knowledge-based Systems (KBS), where the need for methodologies for acquiring, modeling and managing knowledge at the conceptual level has long been accepted. The distinct nature of planning applications clearly distinguishs KEPS from general knowledge-based systems chiefly in the area of acquisition and representation of knowledge about actions (McCluskey and Simpson, 2004).

2.3.1 Domain Modelling

A planning domain model is assumed to be the declarative description of the domain functionalities. The most important part of the domain model is a set of action description. One of the main advantages of the domain model is that a planning agent can make rational decisions to solve problems. The acquisition of a domain model is performed by knowledge engineer which may involve the analysis of the domain, background of the domain from existing documentation, manuals, interviewing to the

stakeholders etc. During the analysis the knowledge engineer seeks answer of some questions such as: Is this a planning problem? How the actions can be modelled? Can the domain be modelled for real-world problems? Any requirement for expected plans? How to optimise the generated plans? Etc. (McCluskey, 2000a).

2.3.1.1 Domain Analysis

The idea of the domain analysis plays a fundamental role to assist the domain modelling language for designing domain models and to find required planning algorithms. The acquired knowledge usually categorised as follows -

- Domain Structure - The domain structure mainly concerns with the acquired relevant objects and classes, predicates, relationships and constraints (if any).
- Domain Dynamics - The dynamics of a domain model are to define the 'truth value' of an object during the execution of an action.
- Domain Heuristics - To define general approximation rules to help desired plan generation.

Once the domain is modelled by using an appropriate modelling language, domain analysis is used to check the consistency of domain model and identify bugs. The analysis can be done by the planning system or can be added through the problem specification or by using an automated knowledge discovery tool like, TIM (Cresswell et al., 2002). TIM domain analysis tool infers the types and invariants form input domain definition and initial states. TIM also analyses domain models by constructing a set of Finite State Machine (FSM) from operators and describes all the possible transitions for any single objects in the domain.

Generally, the output of domain analysis is the domain knowledge which can be found from the domain specification. Domain analysis can help the planning system and knowledge engineering in the following ways (McCluskey, 2000a)-

- Speed-up Planning
- Improve Plan Quality
- Model Validation
- Match Planning Technology with Domain

2.3.1.2 Planning Domain Design Process

The design process is one of the important elements to the success of developing and maintaining real-world planning applications. To the best of our knowledge there is no standard Planning Domain Design Process to follow by knowledge engineers. In this section we discuss a semi-standard planning domain design process which was derived by (Vaquero et al., 2011b) from existing research on knowledge engineering for planning and design process from software engineering discipline (Sommerville, 2004). This process is a partial ordered sequence of steps that can be discussed as below -

1. **Knowledge Acquisition** - The knowledge acquisition phase mainly identifies the requirements specification of the domain. Knowledge can be acquired by using expert knowledge, documentation, analysing the system as a whole or by using viewpoint analysis
2. **Knowledge Modelling** - The Knowledge model describes the overall requirements for the planning and scheduling application. This is one of the central

parts of the KE process that ensures the quality of the domain model. It defines a set of problems that a planner needs to solve. It also covers the formal descriptions of actions and their pre- and post conditions, and their nature that can be performed including the types of objects and their relationship.

A knowledge model is needed to be validated by using some metrics such as - *Accurate*, *Adequate*, and *Complete*.

3. **Knowledge Formulation** - Knowledge Model must be formulated in an adequate planning domain definition language, such as PDDL, NDDL, OCL etc. to generate plan using appropriate planner.

4. **Knowledge Validation** - Formulated knowledge must be tested. The test can performed by static testing to identify bugs earlier and dynamic testing by using a planner.

   (a) Static testing - Does not require any planner or problem definition. This will check the syntax, semantics, objects relationship and consistency.

   (b) Dynamic testing - Dynamic testing requires the a complete domain model and a problem definition.

   (c) Plan Generation - By using an iterative process of knowledge validation, one or more planner can generate a plan (if there is any).

5. **Plan Analysis and Post Design** - To ensure that the generated plan is valid according to the knowledge requirement using some metrics. This can be performed by using VAL (Howey et al., 2004) which validates actions execution as well as recognise whether the plan reached to the goal.

2.4 Domain Modelling Languages

The control mechanism of Automated Planning needs to be able to represent and reason with rich, expressive and detailed knowledge of such phenomena as movement and resource consumption in the context of uncertain and continuously changing environmental conditions (Bresina et al., 2002). Traditionally, domain definition is a physical system with discrete and continuously-varying aspects have been represented using the mathematical notion of a hybrid dynamical system. This is a system that has state made up of a set of real and discrete-valued variables that change over time according to some fixed set of constraints. Hybrid systems are used for modelling of applications such as embedded control systems (Carloni et al., 2006). In general, the more expressive representation languages used to model the problem world, the harder to find a planning algorithm to solve problem expressed in the input language, and the speed of the resulting algorithm decreases as well (Wilkins, 1988). Therefore, practical planning systems must make enough restricting assumptions so that a viable, efficient implementation can still be realised.

In order to represent the domain knowledge in some representation language the language should have the following attributes (McCluskey, 2000a) -

Structured: The representation language should be well-structured to acquire complex actions, states and broken down complex objects into manageable and meaningful units.

Associated Workflow: The language should follow a set of steps to acquire a domain model to help knowledge engineering processes.

Support in Operation: The language should provide some metrics to evaluate the

developed model.

Support with tools: It should be tool supported to provide static testing and check the operationallity.

Expressive: The language should be expressive to capture complex aspect of real-world problems.

Syntax and Semantics: The language should have clear syntactic structure and meaningful semantics to evaluate with some well-known formal languages.

There are a great number of languages used by the planning community such as PDDL, OCL, HPDL, ANML, NDDL, RDDL etc. Also, each of the languages has many versions available according to the expressivity. The HPDL (Hierarchical Planning Domain Language)(Castillo et al., 2006) is the extension of PDDL2.1 (Fox and Long, 2001) to adapt HTN planning. It is structurally different from the PDDL because of its task hierarchy. HPDL manages the temporal constraints by multi task ordering, time annotations, durative inference task, time initial literals and temporal landmarks. HPDL allows multiple task ordering schemas in sequential order, parallel order or undefined order. In the sequential order the tasks are carried in the same order according to the initial declaration. Tasks or actions are executed simultaneously, i.e., two tasks start at the same time and continuous. Also, tasks can be carried out in any order to achieve the goal.

Model based languages such as ANML and NDDL are used by NASA for Mars Rover Application Using KE tool called EUROPA (Barreiro et al., 2012). The New Domain Definition Language (NDDL), introduced by NASA, is a model-based domain definition language that supports the specification of the concurrent transition system

models (Muscettola et al., 1998). Both ANML and NDDL are based on the AML and PDDL to represents action and state, supports temporal constraints, and provides clear expression for pre- and post-condition for action representation (Smith et al., 2008).

In this section we discus two domain modelling languages called PDDL and OCL to evaluate knowledge engineering methods.

2.4.1 STRIPS Based Language

STRIPS formulations (Fikes and Nilsson, 1971a) are the most widely used planning representation languages in the planning community. In a STRIPS system, actions are described in terms of preconditions and effects, as add and delete lists. Currently, the STRIPS based representation languages are dominating the planning research. In the following subsection, we discuss the standard STRIPS based planning language and its variations.

2.4.1.1 PDDL

AI planning research evolved a standard domain definition language in planning is PDDL (planning domain description language), which is based around a world view of parameterised actions and states, where it is assumed that a planner generates a collection of instantiated actions to solve some goal posed as state conditions. PDDL was introduced by Drew McDermott in the AIPS-98 competition to have a single common representation language for defining domain models. The competitors used different planners to solve the same problems written in PDDL domain (AIPS-98 Planning Competition Committee, 1998). The first version of PDDL is 1.2 is the base of all the PDDL variants as it uses the basic representation of literals, a set of operator schemas

to represent domain actions as classical planning (Fikes and Nilsson, 1971a). The operator definition in PDDL is same as the STRIPS-style uses pre-condition and effect.

Since the inception of PDDL, it has been developed and extended by many researchers. The significant extension for the expressiveness is the PDDL2.1 (Fox and Long, 2001), the official language of IPC-3, that supports numeric fluents and durative actions. Following that extension the language has progressed to PDDL2.2 (Edelkamp and Hoffmann, 2004) to introduce derived predicates, PDDL3.0 (Gerevini and Long, 2005) to introduce durative predicates and PDDL3.1 is the standard language for IPC 6 and 7 in the deterministic track to introduce object-fluents. It has been extended to cope with real applications such as crisis management (Fernández-Olivares et al., 2006) and workflow generation (Riabov and Liu, 2006), and has versions which can represent time and resources. More expressive modelling languages such as PDDL+ (Howey et al., 2004) have been developed for applications where reasoning about processes and events in a hybrid discrete or continuous world is necessary (Fox and Long, 2006). PDDL+ was recently used in an application for developing multiple battery usage policies (Fox et al., 2011). Although PDDL is designed for logical precondition achievement, specialist forms of planning can be incorporated into the language using procedural attachment (Eyerich et al., 2010).

A typical planning problem for PDDL domain includes *domain model* and *problem definition*. The domain model defines the overall structure of the world, for example, types of objects in the domain, kinds of actions and precondition and effects of those actions. The problem definition defines the task that a planner need to perform to achieve some goals with respect to the initial and goal state defined for the considered domain.

2.4.2 HTN Based Language

HTN uses the hierarchical abstract schemas to represent actions and states similar to the STRIPS style planning. HTN represents the level of abstraction of the planning problem and defines the solutions in the application. The solution of an HTN planning system is the decomposition of the abstract task to achieve the primitive tasks as the steps of the plan. In HTN based languages, it is common to order the variables to satisfy given condition of actions.

2.4.2.1 Object Centred Language (OCL)

The object-centred planning approaches by McCluskey and Porteous (McCluskey and Porteous, 1997) is a tool supported planning domain definition language for acquiring the knowledge of the world as domain model in order to implement with suitable planning algorithm. This is a structured language that allows knowledge engineers to develop the segment of the world and validate with the supporting tool. This approach has been supported by different planning methods like partial or total order planning (Kitchin, 2000), graphplan (Simpson et al., 2000), HTN plan (McCluskey, 2000b), and continuous planning with *OCLPlus* (Shah et al., 2012). In the project PLANFORM, a Graphical Interface for Planning with Objects (GIPO) (Simpson et al., 2001) was developed to help the application users to construct, analyse and test the planning domain, which makes OCL_h (McCluskey and Kitchin, 1998) suitable for real applications.

OCL_h is a family language of *OCL* which is based on the idea of engineering a planning domain so that the universe of potential states of objects can be defined first, before operator definition (McCluskey and Porteous, 1997). Developing a domain model in *OCL* uses the following steps -

- Sorts - Identify class and hierarchy,
- Objects - Identify objects of the classes identified,
- Atomic invariants - Static properties of domain model which will not be changed during execution,
- Substate class expression - define object behaviour,
- Operators - precondition and effect,
- Methods - compound task are used for HTN type planning,
- Problem instance - define goal and initial states of objects.

An action in an OCL_h domain model is represented by either a primitive operator or a method. Primitive operators specify under what condition objects may go through single transitions; compound actions specify under what condition objects go through a sequence of ordered actions. The action is primitive or compound depending on whether the name is primitive or compound.

Compound actions need further expansion and cannot be executed directly. In OCL_h, compound actions including methods and achieve(Goal) actions. Methods are compound actions that can be expanded further down under certain restrictions. By eliminating the lower level tasks and orderings and variable binding that may lead to dead ends, the search space of a method expansion is greatly reduced.

2.5 Tool Supported Knowledge Engineering

Traditionally knowledge engineering for automated planning is carried out by using the hand coding method (using a text editor) but to create a non-trivial domain model requires tool support. The tool support depends on the size of the application. Generally, hand-coded domain models are debugged by simple syntax checkers or dynamic testing. The dynamic testing shows the domain errors, does not generate any plan or executes any plan which is inefficient for large real-world applications (McCluskey, 2000a). In the practical point of view tool supported knowledge engineering helps user to acquire knowledge and validate them without any planner involvement. In most cases the static validation is performed to check consistency and remove bugs of domain model. From the user perspective an ideal knowledge engineering tool (see Figure 2.1) integrates all these facilities in one platform by using some interactive graphical metaphors. Hence, the user will be able to design, develop, test and maintain domain models in a single environment.

The tool support for knowledge engineering for automated planning is a longstanding research area. Some of the most successful planning projects like O-Plan (Tate et al., 1994) and SIPE (Wilkins and Myers, 1994) projects used tool support to assist the knowledge engineering process. The need for the tool support in the KE research leads a number of domain independent and domain-dependent tool such as, GIPO (Simpson et al., 2007), itSIMPLE (Vaquero et al., 2012), JABBAH (González-Ferrer et al., 2009), EUROPA (Barreiro et al., n.d.) etc. A detailed overview of the knowledge engineering tools will be given in the following chapter (Chapter 3).

2.6 Current State of Knowledge Engineering

The area of knowledge engineering is an important area within the automated planning research; especially to formulate real-world domain knowledge. The field has advanced steadily in recent years, helped by a series of ICAPS Competitions on Knowledge Engineering for Planning and Scheduling ICKEPS'05, ICKEPS'07, ICKEPS'09 and ICKEPS'12[1], the build up of experience from planning applications, along with well developed support environments (for example, NASA's *EUROPA* (Barreiro et al., 2012)). This area has also been supported by a series of ICAPS workshops on Knowledge Engineering for Planning and Scheduling (KEPS): KEPS'08, KEPS'10, KEPS'11 and KEPS'13[2].

On the other hand, the PLANET Roadmap (Biundo et al., 2003; McCluskey, 2000a), a technical report on the European Network of Excellence in AI Planning, discussed the issues and challenges of knowledge engineering. The aim of the document was to find an established platform for building rich planning system. Knowledge Engineering (KE) part of the document is also directed to the future knowledge engineering issues. It also suggests an ideal environment for knowledge engineering for planning (see figure 2.1). This idealised KE Environment is inspired by PLANFORM Project (Biundo et al., 2003) which could be modified in different model. The main concern of this conceptual KE environment is to consider aspects about the domain knowledge. In this ideal KE Environment, a Knowledge Engineer acquires domain model by analysing many different sources. A Knowledge Engineer plays multiple role as the software analyst. The domain knowledge can be acquired by reading the system

[1] see http://icaps12.poli.usp.br/icaps12/ickeps

[2] http://icaps13.icaps-conference.org/technical-program/workshop-program/knowledge-engineering-for-planning-and-scheduling-keps/

requirement specification, interviewing to the users and managers, observing the existing model. It is the Knowledge Engineer's job to capture the correct knowledge so that those can be encoded using modelling language. It may also require to do the dynamic validation by a using suitable planning algorithm. The plan that has been generated as a result of dynamic validation can be evaluated by comparing with the existing plan.

2.6.1 ICKEPS Competition

The aim of the ICKEPS competition is to promote the development of Knowledge Engineering tools and environments in order to get accessible, acceptable and effective way of developing reliable planning and scheduling systems (Bartak et al., 2010). ICKEPS aims to improve the whole knowledge engineering process through competition. The ICKEPS competition has set some rules for evaluating the KE tools to find the winner in the competition. The first ICKEPS[1] considered KE tools that fall into any of the following categories -

- **Knowledge formulation:** The goal is to find the KE tool have the capability of knowledge capturing and control heuristics.
- **Planner configuration:** The goal is to find if the tool can integrate third-party planners for plan generation.
- **Validation of the domain model:** The tool must be capable of using visualization to validation.
- **Knowledge refinement and maintenance:** The tool should be capable of refine and maintain knowledge through automated learning/training, or a mixed

[1] http://helios.hud.ac.uk/scomtlm/competition/rules.html

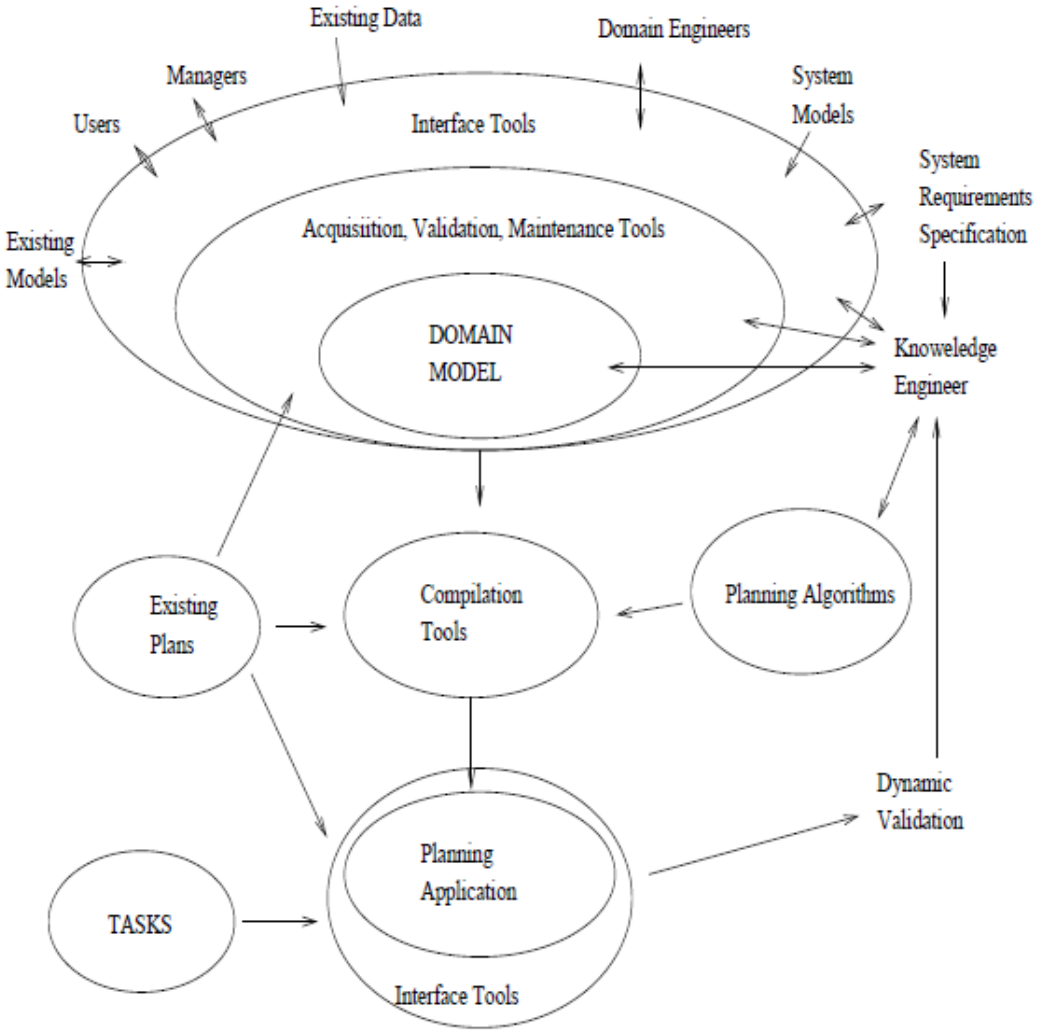

Figure 2.1: An Ideal Knowledge Engineering Environment (Extracted from (Biundo et al., 2003))

initiative P & S process.

The evaluation Criteria used in the first ICKEPS Competition are described below:

- **Support potential:** Does the tool obeys the criteria of the competition? Will the tool be effective to save time than using other method?
- **Scope:** How the tool fits with the scope of the competition?
- **Usability:** Is the tool user oriented? Is this suitable for non-expert user?
- **Interoperability:** Can the tool add third-party planners? Does the user interface of the tool helps the user to design the domain model and test with suitable planning software?
- **Innovation:** Is there any scientific value achieved by the tool?
- **Wider comparison:** Is the tool comparable to other existing KE tools? Can it be compared with other knowledge based system?
- **Build quality:** Is this an well-designed and tested software?
- **Relevance:** Is this tool related to the KE research? Is the tool suitable for designing real-world domain models?

It can be noted that the first two competitions aimed to find tools based on general categories such as 'planner configuration' which is now common to most of the state-of-the-art KE tools. If we see two recent competitions on ICKEPS (3rd and 4th), the evaluation criteria have been narrowed down, focus on particular aspects on the KE tools and techniques. Both (3rd and 4th) competitions were focused on the tools, methods, translation to a particular planning language (if required). The consideration

also given on the application areas, for example, Web Services, Work-flow, Business Process Modelling, E-Learning, Games and Narrative Generation. The KE tools have also been considered if they can use domain-independent planning algorithm to solve basic problems (Bartak et al., 2010).

Due to the nature of competition and a variety of tools and techniques, it is difficult to set any objective measures on 3rd ICKEPS (Bartak et al., 2010). Given these constraints, the judges considered four important issues:

1. User Related Issues
2. Planning and Scheduling Related Issues
3. Software Engineering Issues
4. General Scientific Issues

By considering the above issues a set of detail criteria were used to find the best tool in the competition -

- **Portability:** This is identified as the software engineering issue to find the portability of the software ”difficulty of using the tool out of the laptop of the competitor” (Design Process track).
- **Robustness:** This is also the software engineering issue to find the measurement of how much is the tool sensible to the domain in use (general tools) or how much is the tool sensible to the specific problem in the applicative are (specific tools).
- **Usability:** This is user related issue to find the satisfaction of either by AI experts or target domain experts.

- **Spread of use of the tool:** Mainly to find the usability by finding the number of user used the tool.
- **Perceived added value:** It is important to have some added value int he P&S community as general tools or to the application area as specific tools. The goal is to find the impact of the tool in that particular area of research.
- **Flexibility** Is the tool flexible to use? How easy the user can learn? Can the tool capture complex aspect of the domain?

2.7 Summary

The research challenge arisen in the area of knowledge engineering for planning and scheduling (KE for P & S) from the earlier discussion, is closely related to this thesis. It is clear that knowledge engineering is an important area of research for automated planning and scheduling area. We have discussed the history of automated planning to understand the respective progress of knowledge engineering research. We also discussed the current state of the KE research by the support of ICKEPS competition to promote the research on building KE tool. The evaluation criteria of the competition are illuminating as we can see the progress in this field. Although, state-of-the-art KE tools are quite capable of designing complex domain models but there are still many issues remain unattended. Moreover, we still do not know what is the best way to encode a domain model for real-world applications. The following chapter discusses a range of KE tools for automated planning to find the types of features.

Chapter 3

Knowledge Engineering Tools and Techniques

Knowledge Engineering for Planning and Scheduling (KEPS) is an important research area that deals with the engineering planning domain models. KEPS is defined in the PLANET RoadMap (McCluskey, 2000a) as the process of acquisition, validation and maintenance of planning domain knowledge, and the selection and optimization of appropriate planning machinery to work on them. The main goal of knowledge engineering process is to support the planning system to solve various types of planning problems. A real-world planning application requires suitable tools and techniques depending on the size, nature and complexity of the domain. Naturally, an automated planning system is knowledge-based. Therefore, to develop a planning system, generally it is required to model the knowledge associated with the domain under consideration. These models are called Planning domain models. This knowledge can be encoded by knowledge engineers (usually the planning researchers) by using different techniques. The most common knowledge encoding process is the traditional hand-

coding by using a simple text editor. In the knowledge design process, researchers mainly use basic syntax checker and dynamic testing which is inefficient for the large real-world application. The research in knowledge engineering for automated planning is not mature enough in compare to the other areas to define a standard design process. On the other hand, the planning domain definition languages seem to be very expressive and flexible (e.g. PDDL) to design real-world problems. It is impractical to direct formulation of planning domain models for large real-world application. The lack of user-friendly graphical interfaces for the domain formulation, any standard design process or the gap between the knowledge engineering and planning research limits the knowledge engineering process for designing, developing and maintaining a real planning application.

In this chapter we evaluate types of features used in modelling tools by analysing current knowledge engineering tools and techniques to acquire real-world domain knowledge in order to create a planning application. Also, a general overview will be provided on the traditional hand-coding method for domain modelling for automated planning.

3.1 Knowledge Engineering: Tools and Techniques

This section is dedicated to the discussion of leading knowledge engineering (KE) tools and techniques for automated planning. The discussion emphasis on two general planner-independent tools (GIPO and itSIMPLE) and a traditional technique to model planning domain in order to evaluate in the later chapters.

3.1.1 Hand Coding: a traditional technique for KE

PDDL (Ghallab et al., 1998; McDermott, 1998) is an action-based domain definition language which is inspired by STRIPS (Fikes and Nilsson, 1971b) style planning. The core of the PDDL formalism is for expressing the semantics of domain actions, using pre- and post-conditions to represent actions and effects. This is an ad-hoc process of generating and testing domain model where the knowledge engineers iterate the following steps several times:

- encode requirements,
- analyse the model,
- run a set of planners on several easy problems,
- evaluate the resulting plans (if any), and 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.

Requirements for Domain encoding are generally acquired by knowledge engineer using system engineering methods. Mainly the domain modeller goes through the domain documentation; interview the domain expert or goes through the manual. The domain modeller mainly focuses on how to extract objects, predicates, functions, actions and planning problem. Encoding all these requirements in a text editor requires knowledge engineer to be expert on the language. PDDL is a very expressive language and most planner support only a subset of it. Domain declaration uses requirement such as -

- :requirements - this is to define basic requirements for any PDDL domain model. A domain model may support STRIPS to support all STRIPS functionalities, equality to use predicate '=', typing which consists only STRIPS or durative-action to support numeric values,
- :types - defines objects,
- :predicates - defines relations of objects,
- :functions - to handle the numeric and temporal elements,
- :actions - the operators that contain pre- and post-condition.

Generally, a hand coded domain model has to go through dynamic testing for debugging and plan generation. Such testing is performed by any suitable planner to find plan(if any). Generally planners do not produce any plan if there is any error in the domain model or in the problem file but produce error message for debugging. Also, during the dynamic testing it is hard and time consuming to find bugs. This process has to rely on the knowledge engineer's expertise on the encoding language.

3.1.2 TF Method: HTN modelling tool

This is one of the early approaches of Knowledge engineering with tool support. Task Formalism (TF) method (Tate et al., 1998) includes domain requirement analysis with CORE (COntrolled Requirements Expression) for modelling planning domain for NO-LIN (Tate, 1977) and O-Plan (Tate et al., 1994) systems. Modelling domain in the TF Method follows level oriented approach that requires completing a check-list of sequential activities. These activities include identifying actions, developing actions, identifying conditions effects of those actions, temporal constraints, variable types and

model reusability. The design process of TF is accomplished by using TF Workstation (Tate et al., 1996), a GUI for O-Plan for accommodating domain knowledge description. Domain testing is performed in this system by using O-Plan TF compiler to be used by the core O-Plan planning engine. This is an incremental process to run, debug and modify the domain model. The design process that the TF method follows as -

(i) planning the Development of a Domain Description: to identify all the requirements for the domain with domain expert.

(ii) selecting between Action Expansion and Goal Achievement: to find the definition of either action decomposition or goal achievement i.e. the hierarchical structure of action expansion.

(iii) developing the TF Schemata: to construct details of each level of the hierarchy that defined in the previous section.

3.1.3 itSIMPLE: a leading GUI tool for KE

itSIMPLE (Vaquero et al., 2007, 2012), the winner of International Competition on Knowledge Engineering for Planning & Scheduling (IKEPS)[1], is a method and tools environment that enables knowledge engineers to model a planning domain using the Unified Modelling Language (UML) standards. It is a domain independent GUI tool that allows knowledge engineers to acquire knowledge of the domain during the preliminary design. It also supports a set of languages, tools and techniques to execute the design process. The main function of itSIMPLE is to take the UML's Object Constraint Language (Booch et al., 1997) as input through state machine diagrams, and translate

[1] http://kti.mff.cuni.cz/ bartak/ICKEPS2009/results.html

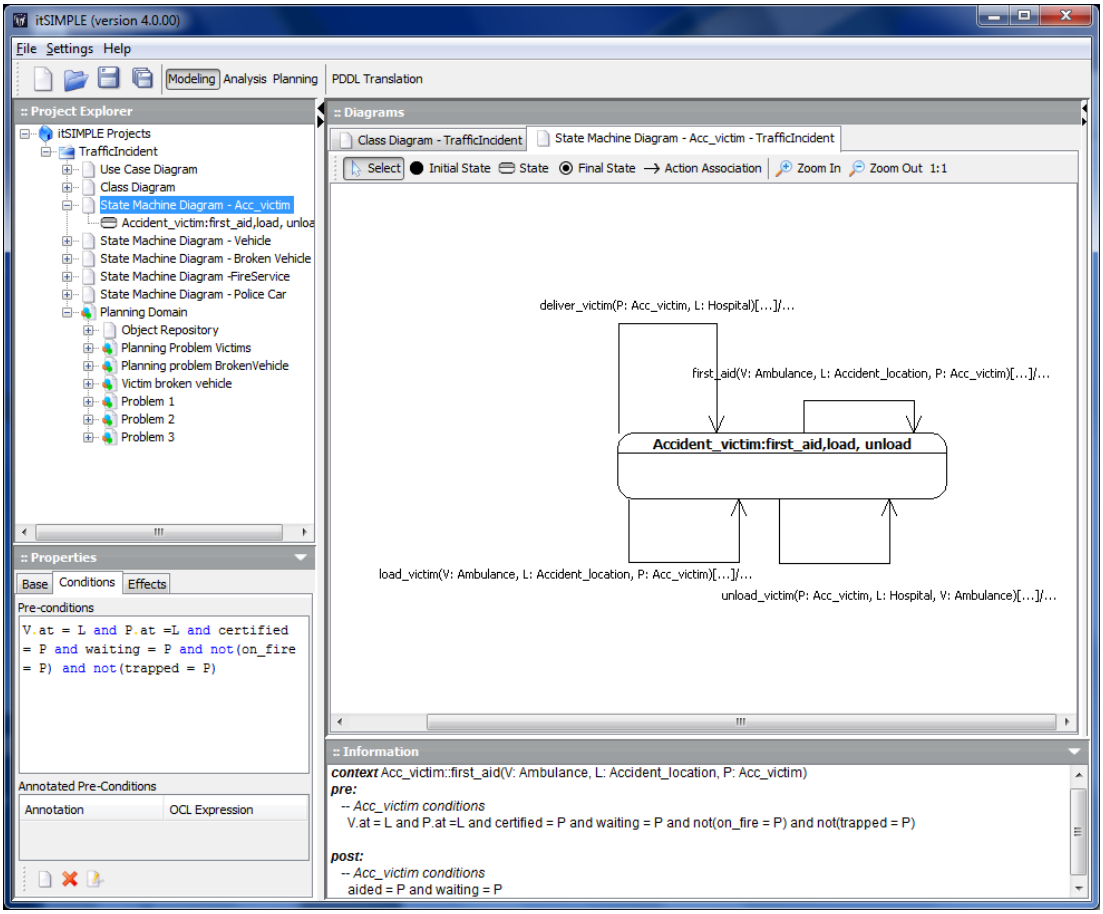

Figure 3.1: itSIMPLE4.0 Domain Modelling Environment

into PDDL. The process is conducted by an intermediate language called eXtensible Markup Language(XML). The user-friendly GUI (see Figure 3.1) of itSIMPLE also helps knowledge engineers to model a domain as object oriented fashion.

This section is dedicated to the analysis of itSIMPLE as a knowledge engineering tool that provides user a flexible GUI support to design a real-world application domain model.

itSIMPLE follows software engineering design processes including UML properties and translating into PDDL model. The general process of designing a domain in itSIMPLE follows steps as -

(i) Use case diagram - This is the preliminary stage of itSIMPLE design process that identifies actors and use-cases for the domain model.

(ii) Design of class diagrams - This is the heart of the whole design process where a number of classes are defined. Classes contain predicates as attributes and operators as methods. Classes are interconnected to each other depending on their relationships.

(iii) Definition of state machines - state machine diagrams are defined for objects and their operators to express pre- and post- conditions. The pre- and post- conditions in the itSIMPLE are expressed in Object Constraint Language expressions (Booch et al., 1997). Also, states are expressed for the possible values of the attributes of classes using conjunctive and disjunctive Object Constraint Language expressions (Vaquero et al., 2009).

(iv) Translation to PDDL - The static information (operator specification) of a domain file are encoded in the UML/XML file during the definition of class and state-machine diagrams. Also this information can be found in the timing diagrams for temporal domain models.

(v) Creating objects - This allows to create as many objects as we want for each of the classes. The itSIMPLE GUI helps to create all the objects by using object diagrams. Objects are stored in the object repository for using in creating problem files.

(vi) Generation of problem files - Problem files in PDDL contains initial and goal states.

(vii) Plan generation - itSIMPLE tool allows the integration of STRIPS style planers for domain model testing.

The itSIMPLE knowledge engineering tool emphasises on the overall life-cycle of a planning application design. It provides the user an integrated design and development environment to accommodate knowledge acquisition in the early stage. Like software engineering itSIMPLE use-case identifies the actors of the systems. With its interactive graphical interface, itSIMPLE allows requirement gathering and analysis for functional and non-functional requirements of a domain. These requirements are acquired by using a most commonly used software modelling language called Unified Modelling Language (UML) (Rumbaugh et al., 2004). The graphical and visual components that provided by the UML make the planning domain definition process more comfortable and facilitate communication and analysis of requirements belonging to different viewpoints.

The UML class diagram in the itSIMPLE represents the static properties of a domain by defining the class, properties, relations and constraints. In a class diagram, operators and their parameters and durative operators are defined as static characteristics of the domain model. The dynamic properties of the domain model such as, the pre- and post-condition of an operator is designed by using a formal language supported by UML called Object Constraint Language.

A state machine diagram in the itSIMPLE tool allows the representation of the possible states of an object. An object can change its state during the execution of a planning action. A state machine diagram is built for each class that has dynamic features that is represented in order to specify pre- and post-conditions of actions. To implement such behaviour, itSIMPLE uses a formal constraint language derived from UML, called Object Constraint Language (Rumbaugh et al., 2004). It is a pure spec-

ification language commonly used for invariants on classes and types to describe pre- and post- conditions on operations and methods, to specify constraints on operations, and to specify derivation rules for attributes for expressing over a UML model.

Action duration is an important feature of PDDL to address real-world problems. itSIMPLE represents time-based planning domains using timing diagrams (Rumbaugh et al., 2004). Such diagrams cover dependencies that could be expressed by parameters such as, time-lines that denote the time spent to perform an action. Currently, time slices are allowed in timing diagrams to define durative-actions (Fox and Long, 2001).

Planning problem instances are modelled by using object diagrams representing relationship of each of the objects in the domain for the particular problem. Initial and goal states are defined by using object diagram. It is possible in itSIMPLE to define initial state of a chosen object so that the planner can operate in order to reach the goal state (Vaquero et al., 2007). itSIMPLE only allow any valid relation of objects that is defined earlier in the class definition by using an object diagram checker. It checks the object relations during the problem definition whether a particular object can be associated with another, given the set of multiplicity and other constraints. The object diagram checker helps the designer to create valid states according to the classes definition.

PDDL2.1 introduces the temporal and numerical constraints which allow knowledge engineers to add duration on the action representation. This feature can also be modelled by itSIMPLE and translated into PDDL by using timing diagram. Timing diagram is used to capture temporal information of a domain by using a time-line. This diagram is connected to the itSIMPLE's state machine diagram to get all the attributes of the object. There are some other temporal features of PDDL such as, *at start*, *overall* and *at end* defines conditions and effects of a variable operation, can be modelled

in itSIMPLE environment.

A domain model requires validation during its design phase to reduce errors, make the model consistent and to achieve correct plan. The domain validation is a processes that verifies, validates, refines and enhances knowledge (Berners-Lee et al., 2001). The validation processes can be static or dynamic.

A static domain validation is mainly focus on the syntax check, debugging, relation between objects, predicate definition, function definition. This test can also find the legal pre- and postcondition of an action from the object and predicate definition. itSIMPLE does not provide any straight forward static validation process during the domain modelling process but it's rich GUI supports designer to design error free model. For example, an object is created from any class that must comply with the class and its association according to the defined multiplicity rule. This helps to avoid inconsistent states during the plan generation. Dynamic validation of a domain model involves the testing actions and their relation i.e. how actions are executed and relates to each other. In most case, dynamic validation is carried out with the planner during generation, although it can be done independent of planner. itSIMPLE invokes Petri Nets (PN) (Murata, 1989), a formal representation of dynamic validation which allows the visualisation of the whole system. Unfortunately, this approach is not fully implemented in itSIMPLE tool.

Most of the automated planning engines require adequate domain model input in a common standard language such as PDDL. itSIMPLE translates domain knowledge from UML to PDDL version up to 3.1. The translated PDDL domain and problem files can be exported as PDDL files to execute with various planning system. itSIMPLE allows the user to integrate many planning engine in it's environment. This facilitate the user to run planner at any stage of domain development. Also, it is possible to

integrate more suitable planner and run all of them together to check performance of planners.

One of the important steps in the domain design is the analysis of the design with respect to the initial requirements. itSIMPLE tool provides visualisation of generated plans and analyse the domain variable. The variable tracker analyses variables of the domain molel and provides quality metrics. itSIMPLE comes with a number of planners (e.g. Metric-FF, FF, SGPlan, MIPS-xxl, LPG-TD, LPG, hspsp, and SATPlan) embedded which allows the user to do the dynamic testing by generating plan. It also uses a visualisation tool for plan analysis called 'movie maker' which captures the responses of planner and executes those plans.

3.1.4 GIPO: an object-based GUI tool

This method is based around the hierarchical, object centred language OCL_h, which is a structured and formal language. The foundation of OCL_h is that the potential state of an object is defined before defining operators (McCluskey and Kitchin, 1998). The planning domain definition languages *OCL* and PDDL are traditionally different from each other in many ways especially in syntax and operator definition. GIPO encapsulates the *OCL* syntax and displays the class hierarchy graphically (see Figure 3.2). Graphical Interface for Planning with Objects (GIPO) is one of the first research tools for building planning domain models that won award in the first International Competition on Knowledge Engineering for Planning and Scheduling (IKEPS)[1]. The user interface of GIPO creates abstract view by hiding the larger parts of *OCL* language details and provides facility to validate, and to identify and remove bugs in the early stage.

[1] http://icaps05.icaps-conference.org/

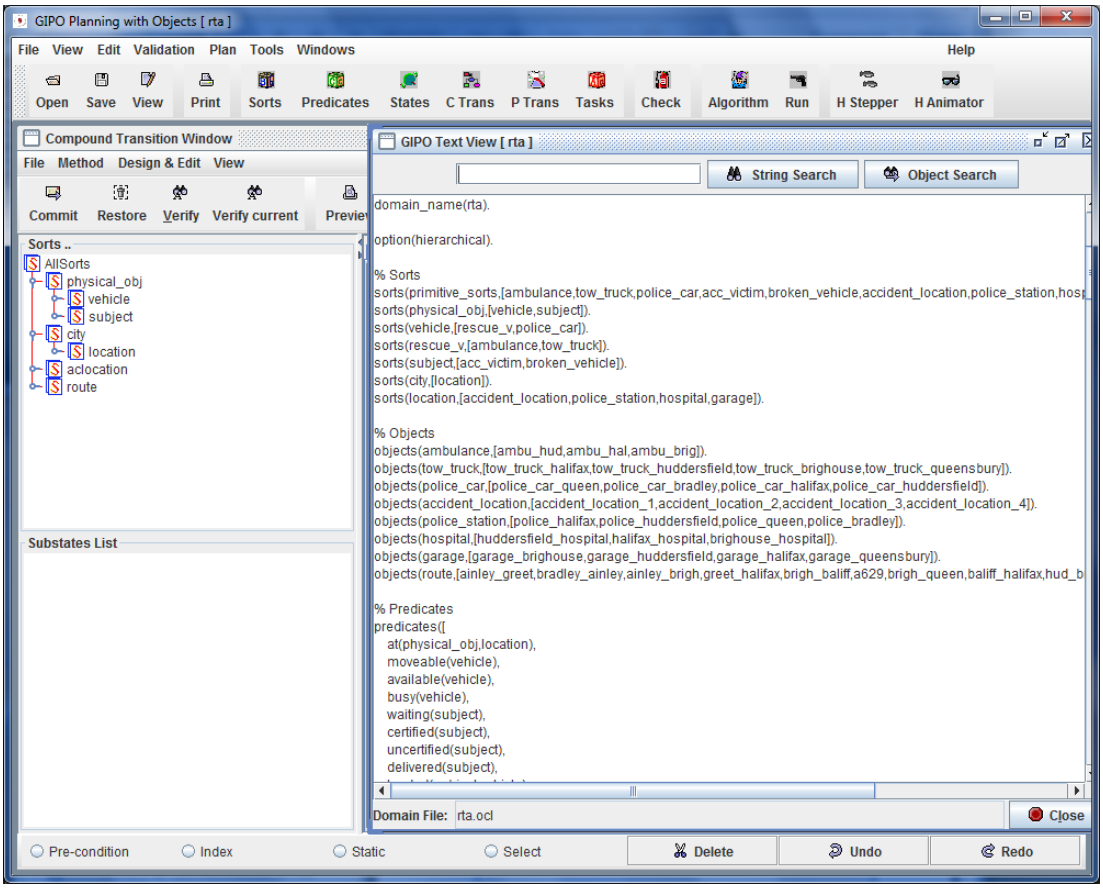

Figure 3.2: GIPO-III: Sort Hierarchy and OCL_h code for RTA Domain Model

It is inspired by formal methods for software engineering, and led to the creation of the knowledge engineering platform GIPO (McCluskey and Simpson, 2006; Simpson et al., 2007). Central to this approach is the precise definition of a planning state as an amalgam of object's individual states. This gives us the concept of a *world state* as one being made up of a set of states of objects, satisfying certain types of constraints. Operator schemas are constrained to be consistent with respect to the state, giving the opportunity for using tools to do consistency checking. GIPO uses a number of consistency checks, e.g. if the object class hierarchy is consistent, object state descriptions satisfies invariants, predicate structures and operator schema are mutually consistent and task specifications are consistent with the domain model. Such consistency checking grantees that GIPO prevents several errors over other method such as hand crafted models.

The steps to design and develop domain model using GIPO are as follows:

(i) Identify a set of *dynamic object classes*, and a class hierarchy, where objects inherit state and behaviour from classes in the hierarchy above it. Thus an object might have superclass mobile and inherit a range of possible states (describing positions) from the mobile class which the mobile object might occupy (hence defining range of behaviour). The object might also be a carrier, and inherit state and behaviour from this also.

(ii) Formally specify constraints on the state of possible objects;

(iii) Develop HTN *methods* in a top down manner, reflecting the main requirements of the domain tasks, followed by development of primitive operators;

(iv) Use static analysis, implemented through tools, to help remove defects from the model as it is developed. Examples are to check operators do not invalidate state

constraints, and tasks obey the *transparency property* (McCluskey and Kitchin, 1998).

(v) Allows the integration of planners, plan generation and visualisation.

GIPO provides a number of editors for each phase in domain construction. The built-in editors allow users to build complete domain model without having any previous knowledge of the representation language *OCL*. It also allows designing and developing flat domain alongside of the hierarchical one. The KE process of GIPO is shown in the Figure 3.3. GIPO allows the user to draw *life history diagram* (basically state machines) that describe the dynamics of the objects of a chosen object class. Thus, the user can define the states that object instances from an object class can take and to show the possible transitions between multiple states. Furthermore, *coordination diagrams* are used to show how objects of two or more concept types coordinate their dynamic movements. Thus, these diagrams allow transitions and states to be linked. While many domain constraints in GIPO are captured graphically, and automatically translated into symbolic representations, others need to be specified textually. This can be done either using the relationships between properties or using more complex predicate calculus expressions to capture the constraints.

Static domain analysis is one of the powerful techniques that make GIPO distinct of its kind. Static analysis checks the syntax, class hierarchy, legal predicates and overall consistency of the domain elements. If the designed domain model passed the check, ready for the dynamic test. The dynamic test can be done by the integrated planners like the *HyHTN* (McCluskey et al., 2003). The plan stepper and animator helps to check plan validation in graphical interface. The steppers work as forward planners to allow users for selecting the actions towards the solution of the problem.

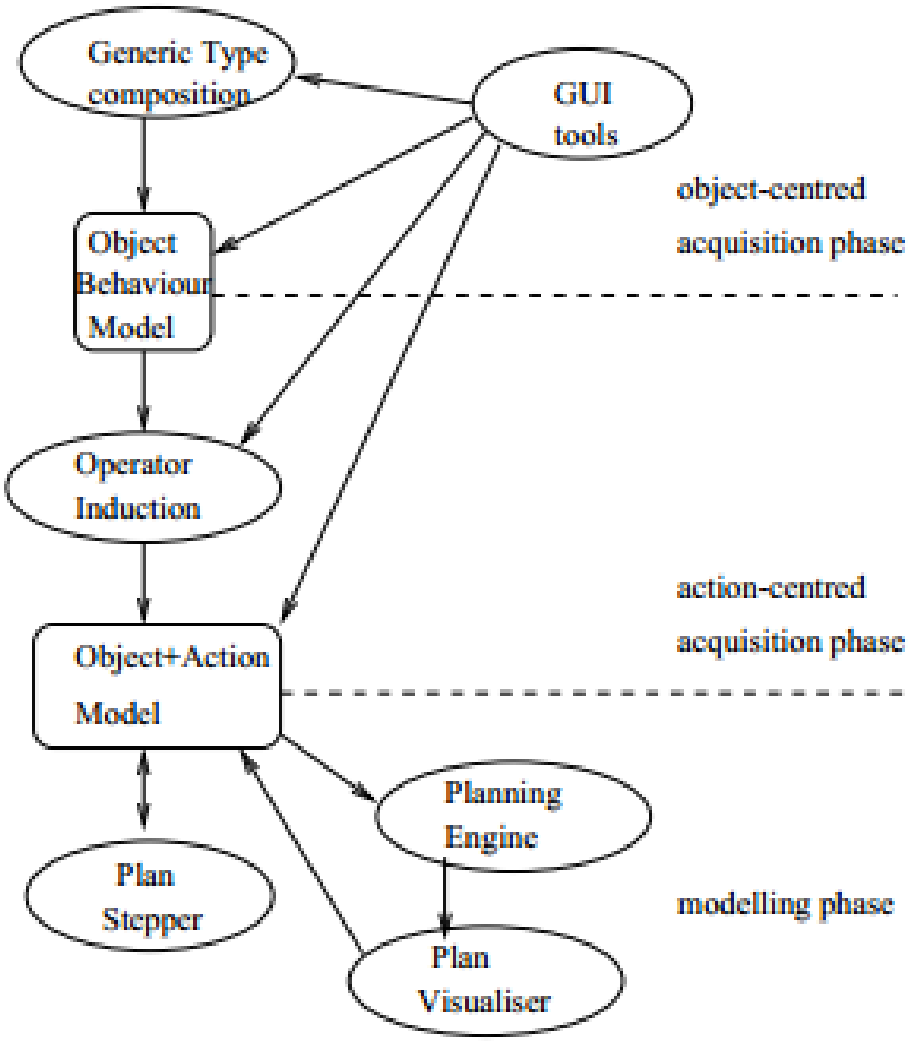

Figure 3.3: Knowledge Engineering Process of GIPO (extracted form (McCluskey and Simpson, 2004))

3.1.5 EUROPA: integrated platform for AI planning

EUROPA (Extensible Universal Remote Operations Planning Architecture) (Barreiro et al., 2012), is an integrated platform for AI planning & Scheduling, constraint programming and optimisation. The main goal of this platform is to deal with complex real-world problem. This has already been used in various space missions on NASA and winner of fourth ICKEPS as the best tool in design process track.

This is an open-source, flexible, user-centred and extensible platform that can be integrated with 'eclipse'[1] IDE. The eclipse plugin provides a number of views for planning and scheduling

- Solver View - One of the important criteria in planning & scheduling is to produce plan. *Europa PSEngine* allows user to create plan solver to execute NDDL script to perform planning task. The solver view allows starting and stopping the *EUROPA* engine, configure the solver and run the solver
- Statistic View - This view provides graphical views of time, decision count, runtime and decision plan.
- Open Decision View - To show how the decision has been taken at each step.
- Schema Browser View - This view can be used if there is an active NDDL model.
- Gantt View - The planning solution can be viewed as a Gantt chart with time
- Details View - A detail plan view is possible by clicking on the Gantt view token.

There are two representation languages, NDDL and ANML (Smith et al., 2008), are used to represent domain knowledge. The representation is like some class libraries

[1] http://www.eclipse.org/

to express the world as object oriented fashion that allows strong data structure for problem instances. This representation also includes the action definition, state of objects, available resources, description of object and plan structure.

Knowledge Engineering is a part of EUROPA's extensive architecture that provides modelling support, result visualisation and an interactive planning process. The modelling supports by Eclipse Plugin to write NDDL code with the help of syntax highlighting. It is parsed as traditional structured programming language compiler. The parser indicates error (if any) as error marker. There is also a text-based editor for supporting ANML language. EUROPA allows ANML models to translate into NDDL and run the translated model with planner. The ANML editor highlights the syntax and associated outline and object hierarchy view.

3.1.6 JABBAH: a domain specific GUI tool

This is the winner of 3rd International Competition on Knowledge Engineering for Planning & Scheduling (ICKEPS) in the track of domain specific tool. JABBAH is an integrated domain-dependent tool that aims to develop process transformation to representing in the corresponding HTN planning domain model. The system mainly deals with the business process and workflow. The processes are represented in terms of Gantt chart (González-Ferrer et al., 2009) or by using open source workflow engine. The tool provides supports for transforming Business Process Management Notation (BPMN) (graphical notation) to HTN-PDDL (Castillo et al., 2006). Such HTN-PDDL domain model is used in HTN planners to obtain steps of solution plan.

3.1.7 VIZ: a Lightweight GUI Tool

VIZ (J. Vodrka, 2010), a light weight knowledge engineering tool, is inspired by GIPO and itSIMPLE. It shares many characteristics of those systems (GIPO and itSIMPLE) in addition to that it provides simple user friendly GUI by allowing naive knowledge engineers to produce PDDL domain models. This tool uses straight forward design process that uses some simple diagrams to produce PDDL domain model. The basic design process in VIZ is as follows -

- *class* definition - this is the same concept to object oriented design to find a set of objects with similar properties.
- *object* - defines instance of a class.
- *variable* - to determine an objects of a class. variables are used to define operators with the help of predicates.
- *predicate* - to define the relations between objects.

VIZ is a simple tool for knowledge engineering produces only simple PDDL output. The tool is under construction and does not allows any third party planner integration.

3.1.8 MARIO: a Goal-Driven application composition tool

Mashup Automation with Runtime Invocation and Orchestration (MARIO) is an integrated framework for composing work flow for multiple platforms such as Web Services and Enterprise Service Bus (Bouillet et al., 2009; Feblowitz et al., 2012). This

tool provides a tag-based knowledge representation language for composition of planning problem and goals. It also provides a web-based GUI for AI planning system so that user can provide software composition goals, views and generate flow with parameter to deploy them into other platform.

3.1.9 PDDL Studio: simple PDDL editor

PDDL Studio (Tomas Plch, 2012), a new PDDL editor has been presented in the ICAPS12 System Demonstration that allows the user to write, and edit PDDL domain and problem files. The main goal of the tool is to provide knowledge engineers' to edit and inspect PDDL codes, regardless of how they were created. The main features of the tool is to -

- Identify syntactic and semantic error
- Highlights the PDDL components
- Planner integration

Like other GUI tool, PDDL Studio does not require to draw any diagram, it is more likely writing traditional programming language code by using Integrated Development Environment (IDE). The current version of this tool can help editing basic PDDL1.7 with error checking.

3.1.10 KEEN: KE environment

Knowledge Engineering Environment (KEEN) (Bernardi et al., 2013) mainly concerns with the services of automated Validation and Verification (V & V). It also supports

the classical knowledge engineering features such as, domain definition and refinement. The KEEN system also indicates the potential of the using validation process of domain model, planner and plan. The tool mainly designed for time-line based approach to acquire temporal behaviour of planning system. KEEN is an integrated tool that requires plugin to the Eclipse for editing and highlighting syntax, getting a tree view of the code blocks and getting real-time syntax checking. The main features of KEEN are:

- integrating V & V Services
- domain Validation
- planner Validation
- plan verification and Dynamic Controllability Check
- plan Validation
- plan Controllers Synthesis

3.2 Features of KE Tools and Techniques

The main goal of GUI tools is to provide knowledge engineers a systematic way to reduce modelling time and errors. Having discussion in the above section, it is clear that there are a great number of effort for developing domain modelling tools to support automated planning applications. Tools discussed above have various characteristics including advantages, disadvantages or limitations on designing domain model. A set of features has been identified after carefully analysing all the tools and techniques, which are summarised as below.

3.2.1 General

There is a number of knowledge engineering tools have been discussed in this chapter. The discussion gives a general idea or view of tools with different characteristics. A tool may help to create various types of domain models or particular type. It may support modeler by integrating third party planners to aid the dynamic testing. We have found some interesting features that are commonly found in existing KE tools.

- **Domain Independent:** Domain Independent tools refer to those systems that do not focus on the particular domain or a particular set of domains. Tools like GIPO and itSIMPLE seems to produce different types of domain model. On the other hand, JABBAH is designed to develop domain related to the Business Process Modelling.

- **Planner Independent:** Domain model produced by using some Knowledge Engineering tools may be used by the designated planners; in that case we call them planner dependent tool, otherwise planner independent. Produced domain model by domain independent tools like GIPO, itSIMPLE, VIZ and PDDL Studio, usually accepted by range of third party AI planners.

- **Planner Integration:** Integrating third party planner with the knowledge engineering tool gives the user to have dynamic test on board. itSIMPLE allows integration of many planners form International Planning Competition (IPC) in both Windows and Linux platform.

- **Collaboration:** Knowledge engineering tools and techniques that have been discussed in this chapter do not support collaborative work among knowledge engineers.

- **Statistics:** Statistics of a domain model designed by a KE tool may give summary of number objects, predicates and actions. A statistics of output domain model may help measuring the quality of the domain model or the tool that has been used to create such model. This statistics may help for evaluating the planner performance.

- **Open Source:** Most of the tools are designed and developed for research purpose and available for the community for further development.

- **Independent to any Operating Systems:** Planners available in itSIMPLE tool are mostly supported by Linux environment. Thus, the tool uses in the Linux operating system gives more support for the modeler to do the wide dynamic testing.

3.2.2 Knowledge Representation

Current state-of-the-art knowledge engineering tools are developed around any particular language. In this subsection we explore the supported languages and features that can be encoded by using current knowledge engineering tools.

- **Language Supported:** PDDL is the standard language in the planning community and a number of planners have been developed to use PDDL domain model. itSIMPLE, PDDL Studio and VIZ are designed for developing PDDL domain model. Main language of GIPO is Object Centred Language (*OCL*) as well as it can translate to PDDL1.2. On the other hand, EUROPA representation languages are NDDL and ANML which are different form PDDL or OCL.

- **Classical Domain:** PDDL Studio allows developing PDDL1.7 which is classical domain and does not express numeric and temporal aspect of planning.
- **Temporal and Numerical Domain:** itSIMPLE tool allows the user to develop many different versions of PDDL domain model including PDDL2.1 to express numeric and temporal constraints in the domain model.
- **Continuous Domain:** Continuous time can be modelled in PDDL+ (Howey et al., 2004) in terms of processes and events. There is no current knowledge engineering tool to develop domain model with such properties.

3.2.3 Debugging and Validation

One of the important reasons of using knowledge engineering tool is the error handling. A hand-coded (using text editor) domain model may contain many errors as it only depends on the dynamic testing.

- **Syntax Error:** GUI tool such as GIPO and itSIMPLE produce domain models by translating from graphical representation to the domain definition languages, usually do not produce any syntactic error. Editing tool like EUROPA and PDDL Studio highlights codes to avoid potential syntax error.
- **Semantic Error:** There could be many semantic problem occur during the domain modelling such as useless type, predicate, operator, irrelevant initial and goal states, unassigned variable, inappropriate class hierarchy and incorrect definition of objects.
- **Static Validation:** The main goal of static validation is to find and remove error in the initial stage of domain development. GIPO's static validation process

checks the class hierarchy, legal predicates and overall domain consistency. it-SIMPLE does not provide any facility of static validation but the UML diagrams such as, class diagram defines the relationship between classes which must be met to build operator transition of in object definition.

- **Dynamic Validation:** Hand-encoded domain models must rely on the dynamic testing although there are some facilities to check syntax by using TIM. Domain independent tool like, itSIMPLE allows a range of planners integrated to perform dynamic validation of a domain model.

- **Debugging:** Most of the tools have some kind of debugging, such as GIPO's static testing, itSIMPLE's UML design and PDDLStudio's error highlighting. To the best of our knowledge there is no complete knowledge engineering tool that provides debugging facility to make a bug free domain model.

3.2.4 Design Efficiency

Although, the trend of domain modelling is using hand-encoding method but it is generally accepted that the tool support will give more flexibility of design experience.

- **Requirement Analysis:** itSIMPLE requires use-cases to find actors and actions, class diagram to define predicates and actions and state machine diagram to create pre- and post- condition for the defined action. Requirement analysis helps the design process to be consistent while creating and using objects. Simple tool like PDDLStudio does not embed the requirement in the tool but the expertise on the domain and the language.

- **Design Process:** A domain model design process involves some organised steps

to help the modeller without being mastering the modelling language. Most of the KE tools support software engineering design process. For example, it-SIMPLE tool supports a semi-standard design process that is discussed in the previous chapter.

- **GUI Support:** Most of the tools provide user-friendly GUI support for designing and developing domain model.

- **Design Speed and Acceptability:** It is assumed that tool support speeds up the process of designing complex domain model compare to the hand-encoding. Generally the using modelling tool like itSIMPLE, forces the modeller to produce Use-case, class and state machine diagram to produce simple domain model like blocks-world. Domain like 'Blocks World' consists of only few operators like *take* and *put* could be created significantly less time than the use of tool. On the other hand, tool can help to create complex problem files with minimum error than the hand coded.

- **User Experience:** Most of the knowledge engineering tools are design concentrating on the researcher who are already expert on the area or on the language. There is no tool that has been designed for the inexperienced user.

3.2.5 Maintenance

Maintenance is one of the important issues in software engineering, which is expensive, time-consuming and challenging.

- **Reverse Engineering:** Most of the GUI tools require diagrammatic representation of the domain in order to translate the diagrams to the required planning

language. For example, itSIMPLE requires class and state machine diagram and GIPO requires Object Life History (OLH) diagram to get PDDL and OCL domain model respectively. Once the model (PDDL or OCL) has been generated, if it is modified outside the tool environment, it is in general not possible to reverse engineer into the environment.

- **Change Handling:** Designing a domain model is a complex process that requires many iterations of testing to debug. During the design process, it always requires changes in many part of the domain. In a domain model, if there is any change requires in any part of the model, tools cannot handle those changes automatically. The modeller needs to change related parts of that domain model.

3.2.6 Operationality

This subsection discusses some operational features of the domain model developed by using tool. Designed and developed domain model by using tools may perform better than the other method like hand-encoding or vice-versa.

- **Model Performance:** There is no published work shows that domain model produced by one method is better than other. It could depend on the complexity of the domain or the experience of the modeller on the particular language.

- **Model Efficiency:** Although the hand-coding method is one of the most exploited method but there is no evidence that shows that domain model produced in this method is more efficient to use than others. Efficiency of the model entirely depends on the expertise of the modeller's expertise on the language.

3.2.7 Support:

Most of the KE tools come with complete documentation including description of the tool, tutorial, updates and a FAQ section. It is easy to find information and solutions for most of the common issues.

3.3 Summary

This chapter presents a range of state-of-the-art knowledge engineering tools for encoding planning domain models. This discussion leads to some interesting features that are present in the current tools. Also there are some features that have been identified that are not common in any of the current tools discussed. Features of the tools are summarised in the Table 3.1. Also, some of the tools are designed for developing specific domain model. There are some interesting features identified that could be added to other tool to get better performance. Although, there are a great number of tools available for planning domain modelling, there are no set guidelines present to evaluate the performance of these tools and techniques. It opens a new research area of developing knowledge engineering techniques for automated planning in order to bridge the gap. Although, there is a number of domain modelling tools available, there is no proof or direction to find the suitable tools or techniques for knowledge engineering.

Features	itSIMPLE	GIPO	EUROPA	JABBAH	VIZ	MARIO	PDDL Studio	KEEN
General								
Domain Independent	YES	YES	YES	NO	YES	NO	YES	YES
Planner Independent	YES	YES	NO	NO	YES	NO	YES	YES
Planner Integration	YES	YES	NO	NO	NO	YES	YES	NO
Collaboration	NO	NO	NO	NO	NO	NO	NO	NO
Open Source	YES	YES	NO	NO	YES	NO	YES	YES
OS Independent	Most of the tools are on Linux based OS. Some of them, like GIPO, itSIMPLE runs on Windows.							
Knowledge Representation								
Language Supported	PDDL	OCL & PDDL	ANML & NDDL	HTN-PDDL	PDDL	Cascade	PDDL	DDL
Classical Domain	YES	YES	NO	YES	YES	NO	YES	YES
Temporal & Numeric Domain	YES	NO	YES	YES	NO	YES	NO	NO
Continuous Domain	None of them is capable.							
Debugging and Validation								
Static Validation	NO	YES	NO	NO	NO	NO	NO	NO
Dynamic Validation	YES	YES	YES	YES	YES	YES	YES	YES
Design Efficiency								
Requirement Analysis	YES	NO	NO					
Design Process	Each of them uses different design process.							
GUI Support	YES	YES	YES	YES	YES	YES	YES	YES
Design Speed and Acceptability	Unknown							
User Experience	Unknown							
Maintenance								
Reverse Engineering	NO	NO	NO	NO	NO	NO	NO	NO
Operationality								
Model Performance	Unknown							
Model Efficiency	Unknown							
Support								
Support	YES	YES	YES	YES	YES	YES	YES	YES

Table 3.1: Types of Features that are identified in the state-of-the-art KE tools

Chapter 4

The Road Traffic Accident Domain

Road traffic accidents are unexpected events that increase congestion and travel time. Traffic accidents reduce road capacity and increase travel time for the road users which leads to more serious concerns such as drivers frustration and cause further accidents. The key to handle such problems lies in systematic planning with the coordination of human and technical resources in the traffic network. This process improves safety of the victims and other road users, which allows traffic agencies to increase the accident management activities safely and efficiently. The characteristics of this area are that goals must be posed and plans must be output in real time. The domain is complex, with road topology, information distribution, traffic flows, driver behaviour. Highways Agencies have to control all these potential factors. The representation and encoding of such domain knowledge, of possible actions and plans, and of potential tasks for the road traffic accident scenario is thus a crucial but difficult issue.

This chapter is concerned with the representation and conceptualisation of road traffic domain in order to assist automated planning to manage traffic accidents. The main goal of this domain is to evaluate knowledge engineering tools and techniques

for building a real world domain model. The domain is interesting as it contains various road topologies, different types of vehicles, information distribution and highways agencies regulations. Moreover, accidents in the road vary from each other and require special attention and planning.

4.1 Road Traffic Accident

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. In particular: the amount of data available to a traffic control-center is already enormous and continues to grow, the amount of data available to travellers is increasing in volume and accuracy. On the other hand, the level of service in terms of factors such as safety, economy, sustainability, ecology, is complex to monitor and optimise. The need to look to reducing costs, while maintaining level of service is a high priority. Costs include the training and maintenance of human expertise in traffic management, as well as the acquisition, configuration and maintenance of software. Current leading edge software to help in traffic management rests primarily on advances in data integration. These advances have been supported in the UK by the common data interface delivered by the Urban Traffic Management and Control (UTMC)[1]. Data integration is leading to enhanced capabilities in the current generation of software, such as the integration of control of traffic management and traveller information systems, but more importantly can be used as a platform for the deployment of more intelligent software services. Accidents cause traffic conges-

[1] http://www.utmc.uk.com/

tion, injury, increase environment pollution, and cost millions of pounds every year for delay and damage. So, highway agencies need more appropriate solution to manage accident promptly. Accidents are a particular type of road traffic incident which is defined as unexpected and unplanned events that impose negative impacts on the road capacity, congestion and travel time (Owens et al., 2000).

4.1.1 Necessity for Accident Management Problem

In England the Highways Agency is responsible for managing, maintaining and implementing strategic road network including all motorways and significant trunk roads. Any incident that disrupts the normal operation of road traffic network is considered as 'National Incident' under the Civil Contingencies Act 2004 as a Category 2 responder (HA, 2009). The main goals of the Highways Agencies to accident management are to:

- improve road safety;
- reduce congestion;
- improve reliability of road network;
- improve accident management and roadworks;
- improve information management on traffic incidents;
- reduce injuries and death of people for traffic accident.

To achieve these goals, the Agencies require effective and efficient standardised 'Command and Control' system that allows the co-ordination between the emergency services and responsible organisations.

Utilising automated planning capabilities in real applications are a current topic with great potential to help with speed, accuracy, and co-ordination of tasks to be carried out. Automated Planning and Scheduling techniques have exposed to perform very efficiently in different types of crises by providing effective plans and strategies either in military crisis (Currie et al., 1991) domains in civil domains oil spills (Bienkowski, 1995), floods (Biundo and Schattenberg, 2001) or forest fires (de la Asunción et al., 2005; Fernández-Olivares et al., 2006). An application for automated planning could help the transport system to generate plans and courses of action in real-time to enable more effective control traffic incidents.

The RTA domain deals with a situation which rises 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 on the accident scene. Once victims are released and free of the wreckage, paramedics must give first aid to them, and then load them into ambulances and deliver them to hospitals. Finally, Tow Trucks have to remove damaged cars from the accident scene in order to restore normal traffic condition.

A typical planning task in the RTA domain is about handling accidents, delivering accident victims to hospitals and restoring the situation to the normal order. The initial situation is defined by a road network and accident. The accident is specified by its location, number of vehicles and victims involved. The initial assumption is that the Ambulances are in Hospitals, Police Cars in Police Station, Fire Brigades in Fire Service Stations and Tow Trucks in Garage. The goal is i) to deliver accident victims to hospitals and damaged cars to garages and ii) return Ambulances to their Hospitals, Police patrols to their Police Stations, Fire Brigades to their Fire Stations and Towing trucks to their garages.

4.2 Requirement Analysis for the RTA Domain

This work has been performed in the context of the EU-funded network *Autonomic Road Transportation Management*[1] consisting of both academics and practising transportation engineers. Using contacts through this network, contributions to transport conferences and workshops, and a set of manuals (Benesch, 2011; HA, 2009; Owens et al., 2000) , a set of requirements have been elicited for the RTA planning problem.

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 the major source of knowledge. This identifies the service provider's responsible for dealing with accidents at an operational level, with police leading co-ordination in and around the scene. The phases of an incident management process 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 requires 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.

[1] www.cost-arts.org

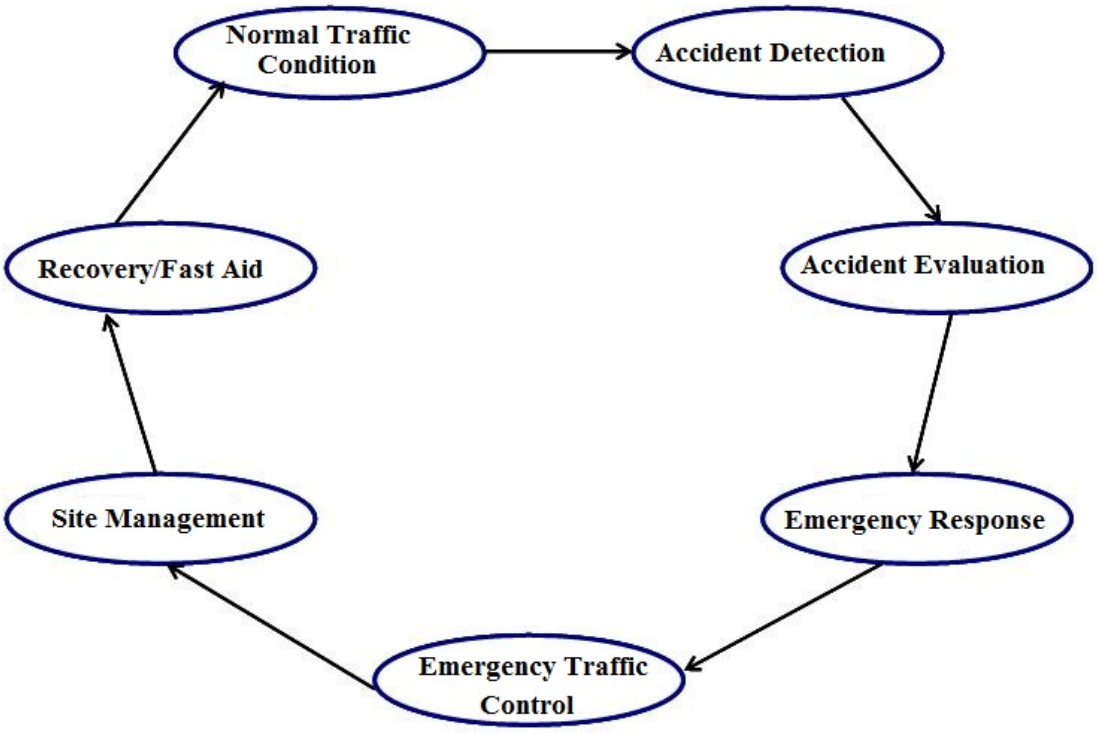

Figure 4.1: Common Accident Management Life-Cycle

The RTA domain is concerned with emergent situations after road accident(s) occur in order to provide accident victims first aid and transport them to hospitals, investigate accident scenes by police and tow damaged cars. The main requirements from a planning point of view are that:

- the accident site has to be secured;
- traffic has to be managed to maintain flow;
- victims have to be taken to hospital; and
- the site has to be cleared off broken cars.

The constraints are that:

- the whole site should be returned to normal (see Figure 4.1) in as short a period as possible;
- victims have to be transported to hospital as soon as possible;
- the state of an accident is not completely known until a police unit arrives.

Contingent on information gathered by police, fire services may have to be involved, if for example there is a fire, or victims are trapped in a car. Given there is only one major contingent point in planning (depending on information gathered by the first agent on the scene) rather than restricting the application to be used with contingent planners, we assume that the contingency can be dealt with either by computing several plans initially, or by re-planning in real time.

4.3 Conceptualisation of the RTA

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

A *Road Network* is represented by 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 6.1), 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 (in contrary to 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 $cap : X \rightarrow \mathbb{N}$ 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 has to confirm an accident or a paramedic has to give a 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).

4.3.1 Defining Operator Families

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 $x \in X_m$ from a location l_1 to a location l_2 ($l_1, l_2 \in V$). As a precondition it must hold that $(l_1, l_2) \in E$ and $loc(x,t) = l_1$. 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. Operators 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 a first aid to accident victims (an ambulance must be on accident scene), certifying an accident by a police (a police car must be on accident scene), or untrapping accident victims by a fire brigade.

4.3.2 Defining Assets and Artefacts for RTA Domain

In this section we describe all the assets (static and mobile), artefacts, attaching and detaching artefacts, and interact in the context of RTA domain.

4.3.2.1 Static Assets:

- *Police stations* - Police cars are kept in the police station. A police officer attend to the accident location to confirm accident and to secure the area.
- *Hospitals* - Accident victims are delivered by ambulances to hospitals for further treatment.
- *Fire service stations* Fire brigade vehicle and other equipments are kept in the fire service station.
- *Routes* - Routes are used to connect one location to another.
- *City Locations* - In the RTA domain model we consider some city location including hospital, police station, fire service station and accident location.
- *Accident locations* - Locations in the city, where accidents happen.

- *Garage* - damaged vehicles are taken to the nearby garage. Tow trucks are also kept in garages.

4.3.2.2 Mobile Assets:

- *Police car* - Police cars are required to confirm the accident in the accident location.
- *Ambulance* - initially kept in hospitals and used to carry accident victims from accident locations to hospitals.
- *Tow truck* - Tow trucks are used to carry damaged vehicles from accident locations to garages.
- *Fire brigades* - Initially, fire brigades are kept in the fire service stations, and used for extinguishing fire and untrapping victims.

4.3.2.3 Artefacts:

- *Accident victims* - Accident victims are mainly human being who could be injured, need first aid or to take to hospital by ambulance.
- *Damaged cars* - damaged car in the accident location could be on fire or stranded. Fire brigade and tow truck are used to extinguishing fire.

4.3.2.4 Attaching and detaching artefacts:

- An artefact accident victim can be carried by ambulance. One ambulance carries only one-accident victim at a time.
- Damaged cars can be carried to the garage by tow trucks.

- An artefact must be free to be attached.
- An artefact must be attached with a mobile asset to be detached.
- Attaching and detaching can not be performed at the same time.
- Rescue vehicles must be free to attach an artefact.
- Attaching and detaching must be performed when the rescue vehicle is on the desired location.

4.3.2.5 Interact: Neither attach nor detach

- A precondition is applied to confirm accident as a police car is required to confirm accident.
- First_aid are given by only in the presence of ambulance.
- Fire brigades are used to extinguish fire and untrap victims.

4.3.3 Operator Description

In this section we describe the identified operators from the families stated above. The operators described in this section must satisfy all the pre- conditions during the planner execution. The table 4.1 summarised PDDL and PDDL2.1 operators and their use in four different domain models.

move: This family operators mainly deals with the movable assets.

- move - This operator mainly deals with the moving vehicle from one city (cluster) to another city (cluster). The precondition is that the vehicle must

List of Operator					
Operator Type	**Operator Name**	**Method A**		**Method B**	
		PDDL	**PDDL2.1**	**PDDL**	**PDDL2.1**
move	move	Y	Y	Y	Y
	move_in_city	Y	Y	Y	Y
attach	load_victim	Y	Y	Y	Y
	load_car	Y	Y	Y	Y
detach	unload_victim	Y	Y	Y	Y
	unload_car	Y	Y	Y	Y
	deliver_vehicle	Y	Y	Y	Y
	deliver_victim	Y	Y	Y	Y
interact	confirm_accident	Y	Y	Y	Y
	first_aid	Y	Y	Y	Y
	extinguish_fire	Y	Y	N	Y
	untrap_victim	Y	Y	N	Y
	start_rescue_victim	N	N	Y	N
	finish_rescue_victim	N	N	Y	N
	start_extinguish_fire	N	N	Y	N
	finish_extinguish_fire	N	N	Y	N

Table 4.1: List of Operators that have been Derived from the Initial Requirement Analysis. In this table 'Y' represents used for the domain model and 'N' represents the operator has not been used for that domain model

be present in the location of origin and another location must be in different city (cluster). Also, route must be connected.

- move_in_city - Vehicles also need to move from one location to another within the same city (cluster). In this case, the vehicle must be available and located in the location of origin.

attach: As we discussed in the previous section that some of the artefacts require to attach to other artefacts to move from one location to another. This family of operators, attaches all the artefacts such as, accident_victims and broken_vehicles to other artefacts such as, ambulance and tow trucks using the following operators.

- load_victim - This operator is executed, if any of the accident victims is injured and required to take hospital. To execute this operator the planner

must check the precondition that the

- vehicle is at the accident location and available
- the victim is certified, first_aided and waiting

In the effect, the victim will be loaded into the ambulance.

- load_car - This operator is used for taking the broken_vehicle to take away (usually in a designated garage). The preconditions are same as the load_victim as both of them are in the same operator family. To take a broken_vehicle to garage the tow truck must be in place before the operator executes.

detach: The artefacts that are attached for the purpose of the move are required to detach once they reach the destination. In this family we have identified four operators named unload_victim, unload_car, deliver_victim and deliver_vehicle as follows -

- unload_victim - This operator comes to play when victims are in the ambulance and the ambulance is at the destination.
- unload_car - Unload car from the tow truck when the tow truck is in the garage.
- deliver_victim - The unloaded victims need to hand over to the hospital by using this operator.
- deliver_vehicle - The unloaded broken vehicle needs to deliver to the garage by using this operator.

interact: The interact operator family plays an important role where interactions are required between artefacts. Operators fall into this family are described below.

- confirm_accident - This is usually the first interact to confirm the accident. Usually, police is required to confirm accident as well as the location.
- first_aid - If there is any victim is injured during the accident the emergency medical service will provide first_aid to the victim before taking to hospital or to any safe place. To provide first_aid the victim must be identified, waiting and should not be trapped in broken vehicle.
- extinguish_fire - This operator is executed if there is any fire in the accident location. The condition is that the fire_brigade vehicle must be in the location if the fire is identified.
- untrap_victim - This operator is used when any victim is trapped in the broken_vehicle. Usually, the fire_brigade vehicle must be present and available in the accident location.
- start_rescue_victim - This operator is used to start the rescuing operation for victim.
- finish_rescue_victim - The rescue process for victims ends by using this method.
- start_extinguish_fire - The fire_brigade starts extinguishing the fire.
- finish_extinguish_fire - once the fire is extinguished this operator will indicate the end of the process.

4.3.4 Function Definition

For temporal domain, a number of functions are used as follows. Values can be defined by the user in the problem files according to the requirement.

- Distance: form location A to location B
- route-length
- confirmation-time
- firstaid-time
- speed
- loading-time
- loading-time-car
- unloading-time
- delivery-time
- untrapping-time
- extinguishing-time

4.4 Summary

This chapter presented a new planning domain model Road Traffic Accidents (RTAs). We discussed the background of the problem, requirements of developing planning application and a conceptual model of the domain. The main goal of this chapter is to assist the following chapters to design, develop and analyse a real-world domain in the context of planning application. The requirements that have been extracted from RTA domain will be implemented in the modelling of the domain. Modelling will be performed by using some popular knowledge engineering tools and techniques, and

will be tested by some suitable planning engine. In the following chapter we develop domain models using these requirements.

Chapter 5

Developing Domain Model

A planning domain model is a formal representation of a planning domain, eventually referred to a real-world application that is encoded in a planning domain definition language. Generally, a planning domain model consists of a set of objects belong to some classes, predicates describing the relation between the objects, a set of operators and constraints that requires to make sense of the world being modelled. A planning problem reflects the problem that can be solved by a suitable planning algorithm by choosing appropriate actions from the domain model. Construction of a domain model is an iterative process that follows a sequence of steps. Since planning domain modelling is an error prone task, especially a real-world domain, requires a reliable method. Using a reliable domain modelling tool or technique for the encoding may save plenty of time of a knowledge engineer and results an almost error free model.

This chapter focuses on the encoding of the Road Traffic Accidents (RTA) management domain. It will be used three well-known knowledge engineering techniques for the comparison and better understanding of the weaknesses and/or strengths of existing KE approaches. The tools and techniques used in this work are as below:

(i) **Method A:** The Traditional Hand-coding Method,

(ii) **Method B:** itSIMPLE - a leading planning GUI, and

(iii) **Method C:** GIPO - an object-based GUI.

5.1 Representation Language

There are various ways to represent domain knowledge by using different domain definition languages. There are a great number of planning domain definition languages available in the planning community. We have used two different domain modelling languages called PDDL and OCL_h. The reasons for using these languages are described in the following subsections.

To encode the Road Traffic Accident (RTA) domain model by using Method A and Method B, we have used the standard domain modelling language PDDL (Ghallab et al., 1998; McDermott, 1998) which is an action-based domain definition language. The PDDL2.1 (Fox and Long, 2001) is more expressive than the traditional one as it supports numeric fluents to encode non-binary resources such as time, energy, distance, speed etc. This version also allows encoding durative actions to encode variables, negative pre-conditions, conditions and effects, non-discrete length etc. Such representations make the language more expressive allowing the solution for real-world problems.

Method C is based on, object centred language OCL_h, is a structured, formal language which mainly deals with the objects of the world. The advantage of using *OCL* is that it can help encoding flat version of the domain model as well as the hierarchical. This is also the default language of GIPO and designed to encode domain knowledge

in more natural structure.

5.2 Method A: The Traditional Hand-coding Method

5.2.1 Overview of the Method A

This is the traditional method of hand-coding a new domain by a PDDL expert, using a text editor and relying on dynamic testing. PDDL domain models can be created by using any text editor such as Gedit, text pad or word pad. A new PDDL (see Appendix A) and PDDL2.1 (see Appendix B) domain is manually encoded using PDDL syntax and tested by using planners like LPG (Gerevini et al., 2003), SGPlan (Chen et al., 2006) and Metric-FF (Hoffmann, 2003).

In the following subsections we discuss the details of creating PDDL and PDDL2.1 domain model and the problem files by explaining this method.

5.2.2 Execution of the Method A

This method (Hand Encoding) can be executed in many different ways depending on the choice of the knowledge engineers. The following steps have been followed for developing RTA domain model according to the requirements acquired in the Chapter 3 -

- **Identify Objects -** Road Traffic Accident (RTA) is the part of a road network that contains roads, cities, locations in the city, a number of vehicles, a number of service vehicles such as Ambulance and Two Trucks and many other objects. The objects in the PDDL domain model are defined in the *types:* as follows -

  (:types

```
    ambulance police_car tow_truck fire_brigade - vehicle
    acc_victim vehicle car - subject
    city_location city - location
    accident_location hospital police_station garage fire_station - city_location
    route
)
```

- **Identify Predicates -** Predicates of the domain model have been defined for the RTA domain as -

```
(:predicates
    (available ?vehicle1 - vehicle)
    (busy ?vehicle1 - vehicle)
    (waiting ?subject1 - subject)
    (certified ?subject1 - subject)
    (aided ?subject1 - acc_victim)
    (connects ?route1 - route ?location1 - location ?location2 - location)
    (route_available ?route1 - route)
    (trapped ?hum - acc_victim)
    (on_fire ?car_acc - car )
  .....
)
```

- **Identify Functions -** Functions are required for PDDL2.1 to express numeric fluents using arithmetic operators in the domain model from primitive numeric expression. In the Hand-encoding PDDL2.1 domain model functions are defined for the RTA domain as in the following expression -

```
(:functions
```

```
    (distance ?O - location ?L - location)
    (route-length ?O - route)
    (confirmation-time)
    (firstaid-time)
    (speed ?V - vehicle)
    (loading-time)
    (loading-time-car)
    (unloading-time)
    (delivery-time)
    (untrapping-time)
    (extinguishing-time)
)
```

- **Identify Operators -** A number of primitive operators have been identified from the requirement analysis to encode the RTA domain model. An example of an operator hand encoded in PDDL2.1 is the *confirm_accident* operator, which is a specific form of the *interact* operator. When an accident happens, a police car must go to accident locations and certifies the *artefacts* involved. The time duration of this action is the so-called confirmation time, which is fixed as one minute per each subject. The code shown in the following example which corresponds to the PDDL2.1 *confirm_accident* operator.

The knowledge requirements and informal description that acquired in Chapter 3, translated directly into PDDL or PDDL2.1, without developing any intermediate notation, using language skill and judgement to perform the encoding. The process is monotonous that leads the model with numerous syntactic error. To model the application a number of diagrams were produced by using pen and paper to find the

```
(:durative-action confirm_accident
  :parameters (?V - police_car ?P - artefact ?A - accident_location)
  :duration (= ?duration (confirmation-time))
  :condition (and
      (at start (at ?V ?A))
      (at start (at ?P ?A))
      (at start (uncertified ?P))
  )
  :effect (and
      (at start (not (uncertified ?P)))
      (at end (waiting ?P))
      (at end (certified ?P))
   ) )
```

Figure 5.1: The durative action *confirm_accident* for RTA domain coded in PDDL2.1

relations of objects, pre- and post-condition of an action, instantiation of variables and the effects of one action on another. As we know in STRIPS style PDDL planning algorithms require two inputs one is the domain and the other one is the problem file. Creating problem files for RTA in a text editor require the clear picture of the domain including the mapping knowledge of the area, actions to perform for particular types of accident according to the requirement. After approximately ten iterations of dynamic testing with planner, simple plans were generated which matched the requirements.

5.2.2.1 Method A: Basic Acceptability Test

Once the Hand-coded domain model has been developed, we carried out a number of test to check the domain models acceptability. The main goal of this test to check the domain model and the problems reflect the requirement of RTA domain.

Test Case PDDL Domain Model: In this test the goal is to take ACC_VICTIM_1 (Accident Victim) to hospital. There are number of hospitals, ambulance and police cars are available to achieve the goal.

Test Result: We did the dynamic testing using LPG planner and the solution found as follows which indicates that the test is passed.

```
0: (MOVE POLICE_CAR_HUDDERSFIELD POLICE_HUDDERSFIELD HUDDERSFIELD
                              ACCIDENT_LOCATION_1 AINLEY_TOP A629) [1]
0: (MOVE AMBU_HUD HUDDERSFIELD_HOSPITAL HUDDERSFIELD
                              ACCIDENT_LOCATION_1 AINLEY_TOP A629) [1]
1: (CONFIRM_ACCIDENT POLICE_CAR_HUDDERSFIELD ACC_VICTIM_1
                                             ACCIDENT_LOCATION_1) [1]
2: (FIRST_AID AMBU_HUD ACC_VICTIM_1 ACCIDENT_LOCATION_1) [1]
3: (LOAD_VICTIM AMBU_HUD ACCIDENT_LOCATION_1 ACC_VICTIM_1) [1]
4: (MOVE AMBU_HUD ACCIDENT_LOCATION_1 AINLEY_TOP HUDDERSFIELD_HOSPITAL
                                                 HUDDERSFIELD A629) [1]
5: (UNLOAD_VICTIM ACC_VICTIM_1 HUDDERSFIELD_HOSPITAL AMBU_HUD) [1]
6: (DELIVER_VICTIM ACC_VICTIM_1 HUDDERSFIELD_HOSPITAL) [1]
```

Test Case PDDL2.1 Domain Model: In the PDDL2.1 test case we use same problem instance to carry accident victim to hospital. In this version we use functions to use duration of an action.

Test Result: The acceptability test is passed for this domain model as shown in the following plan produced by LPG planner.

```
Time: (ACTION) [action Duration; action Cost]
0.0003: (MOVE POLICE_CAR_BRADLEY POLICE_BRADLEY BRADLEY
ACCIDENT_LOCATION_1 AINLEY_TOP BRADLEY_AINLEY) [D:3.0833; C:1.0000]
3.0838: (CONFIRM_ACCIDENT POLICE_CAR_BRADLEY ACC_VICTIM_1
                           ACCIDENT_LOCATION_1) [D:10.0000; C:1.0000]
0.0008: (MOVE AMBU_HUD HUDDERSFIELD_HOSPITAL HUDDERSFIELD
```

```
          ACCIDENT_LOCATION_1 AINLEY_TOP A629) [D:3.3000; C:1.0000]
13.0843: (FIRST_AID AMBU_HUD ACC_VICTIM_1 ACCIDENT_LOCATION_1)
                                                [D:5.0000; C:1.0000]
18.0846: (LOAD_VICTIM AMBU_HUD ACCIDENT_LOCATION_1 ACC_VICTIM_1)
                                                [D:5.0000; C:1.0000]
18.0848: (MOVE AMBU_HUD ACCIDENT_LOCATION_1 AINLEY_TOP
      HUDDERSFIELD_HOSPITAL HUDDERSFIELD A629) [D:3.3000; C:1.0000]
23.0851: (UNLOAD_VICTIM ACC_VICTIM_1 HUDDERSFIELD_HOSPITAL AMBU_HUD)
                                                [D:5.0000; C:1.0000]
28.0853: (DELIVER_VICTIM ACC_VICTIM_1 HUDDERSFIELD_HOSPITAL)
                                                [D:10.0000; C:1.0000]
```

5.3 Method B: itSIMPLE - a leading planning GUI

5.3.1 Overview of the Method B

The main goal of GUI tools is to provide knowledge engineers a systematic way to reduce modelling time and errors. There are a number of tools available to 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 model using the Unified Modelling Language (UML) standards. itSIMPLE, a domain independent GUI tool that allows knowledge engineers to acquire knowledge of the domain in the preliminary design. It also supports a set of languages, tools and techniques to support the design process. The main function of itSIMPLE is to take the UML's Object Con-

straint Language (OCL) as input through state machine diagrams, and translate them into PDDL. The process is conducted by an intermediate language called eXtensible Markup Language(XML). Moreover, the user-friendly GUI on itSIMPLE helps knowledge engineers to model a domain as object oriented (OO) fashion.

5.3.2 Execution of the Method B

The itSIMPLE tool guides the user to a develop domain model by following a semi-standard design process (discussed in chapter 2). The steps in this method, in general to gather the requirements, following the use of UML in software engineering:

(i) use-case diagram

(ii) design of class diagrams

(iii) definition of state machines

(iv) translation to PDDL

(v) generation of problem files

(vi) planning

The use-case diagram is the fundamental of UML design which intends to find related actors and Use-cases related to the system. An example use-case diagram is shown in Figure 5.2 for Ambulance and Tow Truck. The actor Ambulance has the use-cases, such as, move, carry, load and unload victims. The use-case diagrams are the initial design to have better understanding of the actions of any actor from any given domain description. Although, this does not help to translate the UML model but plays a vital role as documentation for future modification.

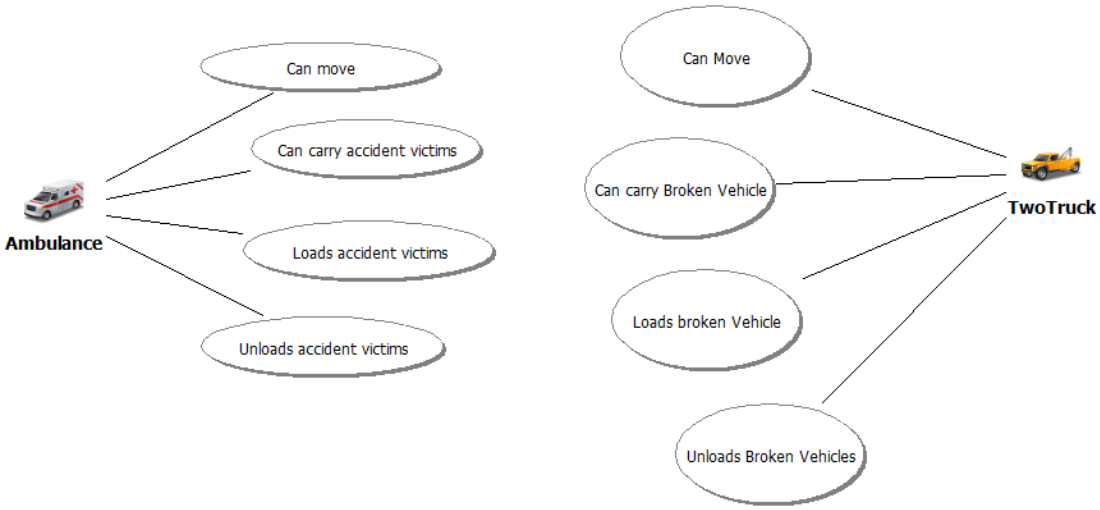

Figure 5.2: Use-case Diagram for Ambulance and Tow Truck

The elements of the class diagram in itSIMPLE represent the static characteristic of the domain. It presents the objects, classes that defines objects, relationships and constraints. RTA domain contains a number of movable and static assets such as, vehicles, subjects, locations and roads, which have been represented in a class diagram 5.3. The class diagram is the heart of itSIMPLE design. In class diagram, the user mainly creates classes and their relationships. The class diagram is like the traditional UML where the user can also create predicates as class attributes and operators as the methods.

Figure 5.4 shows a class example designed in step (ii) for the Vehicle class. Since the Vehicle class refers to *movable assets*, its properties consist of the vehicle's movability, availability or if the vehicle is busy. In the class are also specified the available operators; in the case of a general vehicle, only *move* operators (described in the domain analysis section) are applicable.

While the parameters of the operators and their duration are defined in step (ii), in step (iii) the state machine diagrams help the domain modeller to encode pre- and post-

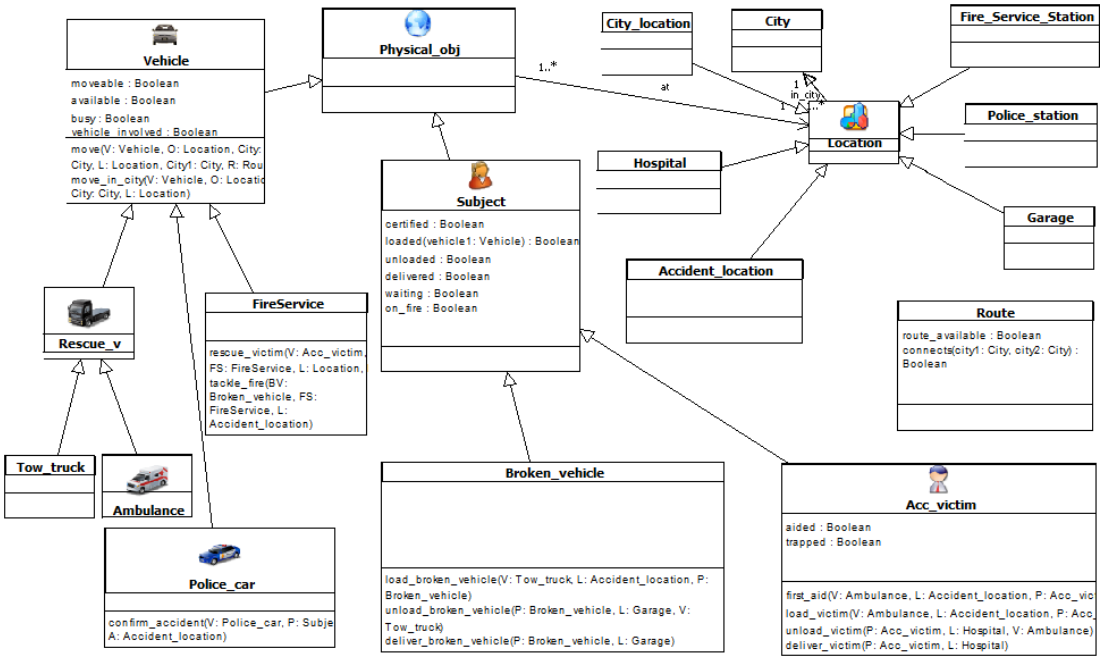

Figure 5.3: Class Diagram of RTA

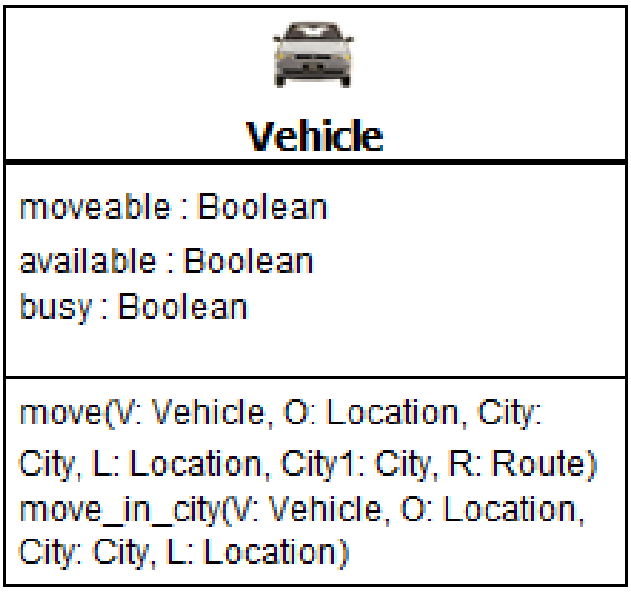

Figure 5.4: Vehicle Class Diagram for Move Operator

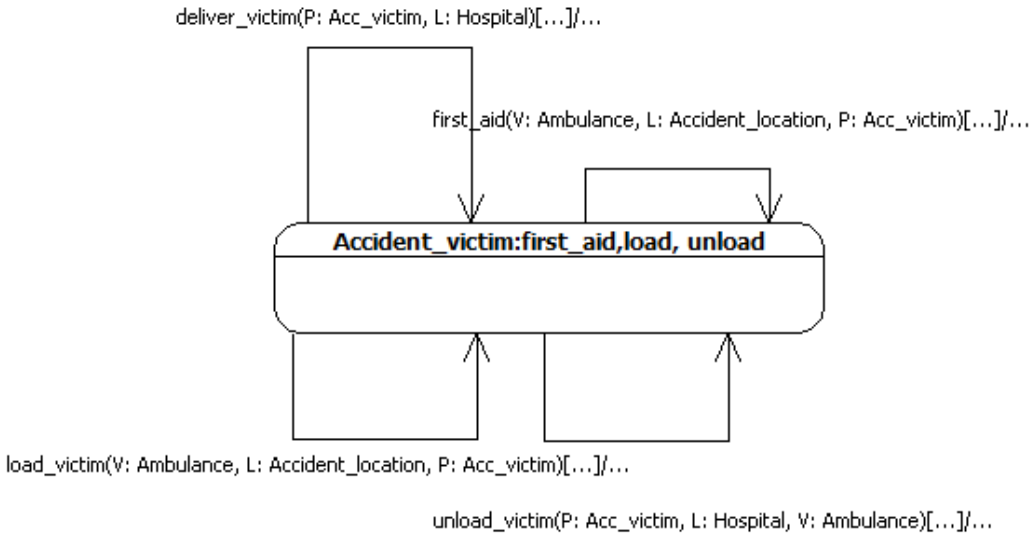

Figure 5.5: State Machine Diagram designed in itSIMPLE for modelling the actions on Accident Victims

condition of operators. The dynamics of the domain is encoded into the domain objects through the operator. Operators and their parameters are defined in the class diagram and the dynamics are represented in the state machine diagram. There are a number of state machine diagrams such as, Acc_victims, Vehicle, Suspect, Broken Vehicle and Police Car have been defined to complete the operators that have been defined during the class definition. In the event of an accident a victim requires to have attended by using a number of actions like -

- *first_aid* - in the presence of a paramedic, carried by ambulance
- *load_victim* - into an ambulance if the ambulance is present in the accident location and available
- *unload_victim* - from ambulance into the location(the location could be a Hospital or any other designated place for the safety of the victim)

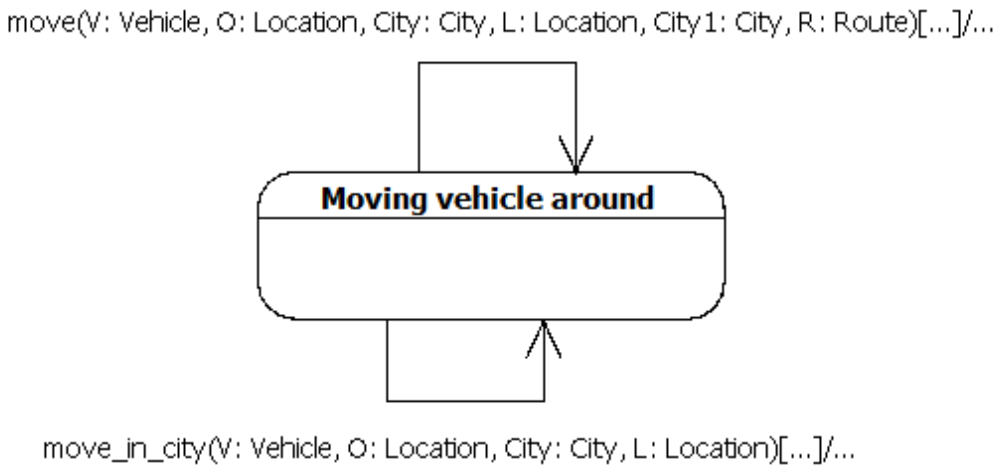

Figure 5.6: State Machine Diagram designed in itSIMPLE for modelling the behaviour of the *Move* Operator

- *deliver_victim* - unloaded victim is delivered to Hospital if all the previous conditions are met

To unload or deliver the accident victims to any hospital, ambulance requires to move from/to the accident location where it has loaded the victims to/from the Hospital. The vehicle is a movable asset and contains *move* action a diagram for the *move* operator that we developed is shown in Figure 5.6. Creating such operator in itSIMPLE requires the knowledge of Object Constraint Language (OCL) (Booch et al., 1997) which is a declarative language can easily associate with UML models. Drawing the state-machine diagrams only give the skeleton of the operators pre- and post- condition.

To create pre- and post- condition of the operator requires the OCL coding as shown respectively in the Figure 5.7 and 5.8. The itSIMPLE translates the diagram into PDDL as shown in the Figure 5.3.2. Similarly, state-machine diagram is required for each of the classes that has defined operators. Another example of state-machine diagram shown in Figure 5.5 has four operators: *first_aid*, *load_victim*, *unload_victim*

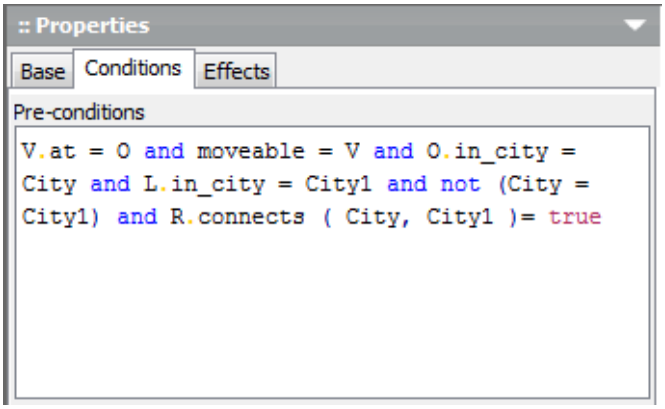

Figure 5.7: Encoding Precondition for *move* Operator

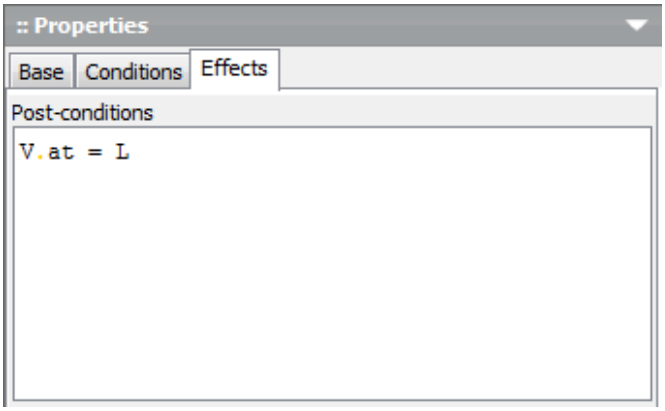

Figure 5.8: Encoding Postcondition for *move* Operator

and *deliver_victim*.

Determining the completeness of the model, in terms of classes and finite state machines, is helped through the use of the structured method. In step (iv), we used itSIMPLE to translate the generated diagrams in both PDDL (see Appendix C) and PDDL2.1 (see Appendix D), 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,

```
(:action move
   :parameters (?V - vehicle ?O - location
   ?City - city ?L - location ?City1 - city
   ?R - road_route)
   :(and
      (at ?V ?O)
        (moveable ?V)
        (in_city ?O ?City)
        (in_city ?L ?City1)
        (connects ?R ?City ?City1)
   )
   :effect
      (and
      (not (at ?V ?O))
         (at ?V ?L)
    )
)
```

Figure 5.9: An action (*move*) for RTA in itSIMPLE

describing their properties in the initial state, and describing the desired properties of objects at the goal state.

An important part of PDDL domain modelling is the definition of problem files. In step (iv), we create problem files in itSIMPLE GUI tool environment by creating a set of objects of each class. In Figure 5.10 an example of a set of objects is shown in the 'Object Repository'. Objects can be used during the creation of the problem file as required. Using itSIMPLE GUI support the user can create as many objects as required. During the creation of objects the user can define the class that the object belongs to. In the problem files, initial and goal states are defined by the association of each object.

The generated domain model and the problem instances are required to be tested by generating plan. itSIMPLE tool comes with a number of third-party planners to test the

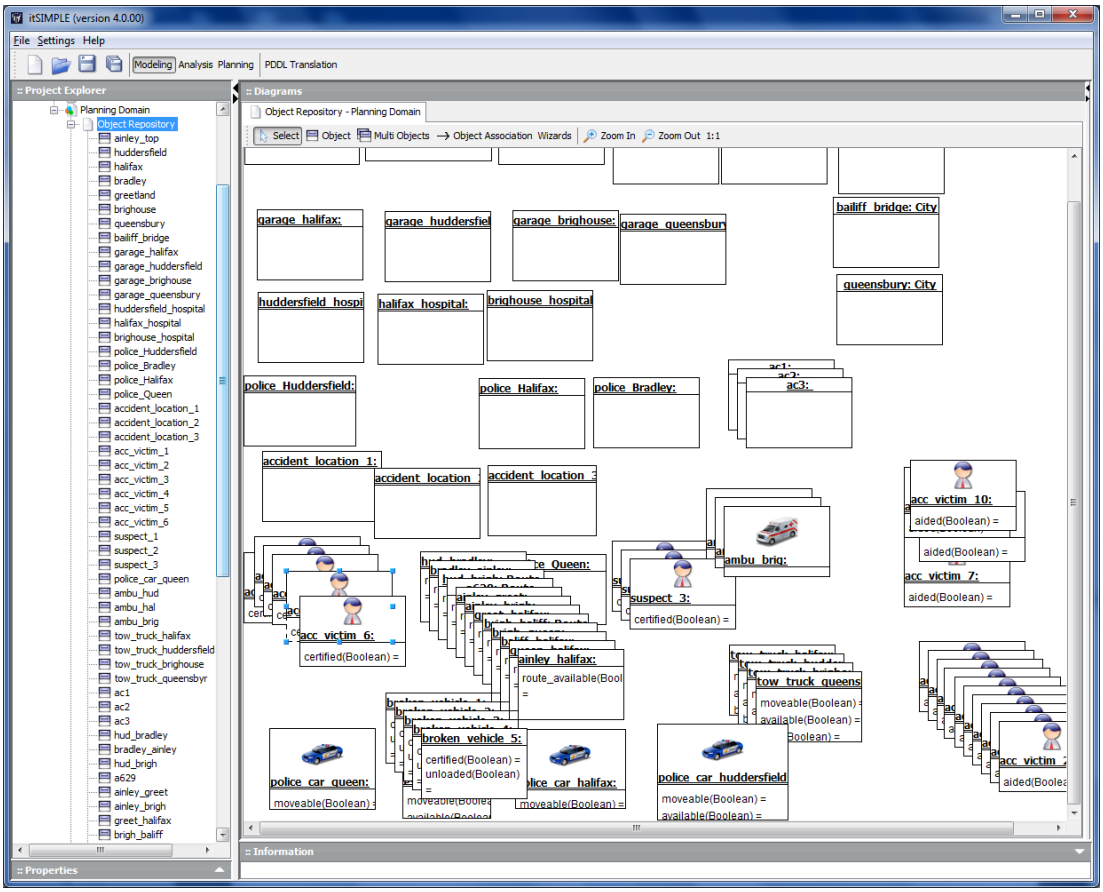

Figure 5.10: Object repository for RTA in itSIPMLE

domain model produced. In step (v), we test the domain models by using compatible planners in order for producing plans and finding errors.

5.3.2.1 Method B: Basic Acceptability Test

In this section we execute two basic acceptability tests for both PDDL and PDDL2.1 domain model. The test has been carried out using different planners randomly. There is an accident happened in the ACCIDENT_LOCATION_1 and some simple task need to carry out to check the domain model's acceptability.

Test Case PDDL Domain Model: The main goal of this test to check that the domain model and problems are working without any error and achieve goals. The goal is to take 'Accident_Victim_1' to hospital. There are a number of Ambulances and hospitals are available so that the planner can choose any of them. The problem was solved by Metric-FF and LPG planners.

Test Result: The following is the test result is from LPG planner's result, which shows that the domain does not contain any error and passed the acceptability test.

```
0: (MOVE POLICE_CAR_HALIFAX POLICE_HALIFAX HALIFAX
        ACCIDENT_LOCATION_1 AINLEY_TOP AINLEY_HALIFAX) [1]
0: (MOVE AMBU_HUD HUDDERSFIELD_HOSPITAL HUDDERSFIELD
                  ACCIDENT_LOCATION_1 AINLEY_TOP A629) [1]
1: (CONFIRM_ACCIDENT POLICE_CAR_HALIFAX ACC_VICTIM_1
                                  ACCIDENT_LOCATION_1) [1]
2: (FIRST_AID AMBU_HUD ACCIDENT_LOCATION_1 ACC_VICTIM_1) [1]
3: (LOAD_VICTIM AMBU_HUD ACCIDENT_LOCATION_1 ACC_VICTIM_1) [1]
4: (MOVE AMBU_HUD ACCIDENT_LOCATION_1 AINLEY_TOP
               HUDDERSFIELD_HOSPITAL HUDDERSFIELD A629) [1]
5: (UNLOAD_VICTIM ACC_VICTIM_1 HUDDERSFIELD_HOSPITAL AMBU_HUD) [1]
```

```
6: (DELIVER_VICTIM ACC_VICTIM_1 HUDDERSFIELD_HOSPITAL) [1]
```

Test Case PDDL2.1 Domain Model: An accident happened in the Accident location 1 and the goal is to take ACC_VICTIM_1 (Accident victim) to hospital.

Test Result: After a number of trials, we found the solution. We check the solution carefully to identify any unusual behaviour. The test is passed which implies that the domain model and the problem instance are acceptable.

```
0.00: (MOVE POLICE_CAR_BRADLEY POLICE_BRADLEY BRADLEY
    ACCIDENT_LOCATION_1 AINLEY_TOP BRADLEY_AINLEY) [3.08]
3.09: (CONFIRM_ACCIDENT POLICE_CAR_BRADLEY ACC_VICTIM_1
                              ACCIDENT_LOCATION_1) [10.00]
13.10: (MOVE AMBU_HAL HALIFAX_HOSPITAL HALIFAX
     ACCIDENT_LOCATION_1 AINLEY_TOP AINLEY_HALIFAX) [5.90]
19.01: (FIRST_AID AMBU_HAL ACCIDENT_LOCATION_1
                                      ACC_VICTIM_1) [5.00]
24.02: (LOAD_VICTIM AMBU_HAL ACCIDENT_LOCATION_1
                                      ACC_VICTIM_1) [5.00]
29.03: (MOVE AMBU_HAL ACCIDENT_LOCATION_1 AINLEY_TOP
           HALIFAX_HOSPITAL HALIFAX AINLEY_HALIFAX) [5.90]
34.94: (UNLOAD_VICTIM ACC_VICTIM_1 HALIFAX_HOSPITAL
                                          AMBU_HAL) [5.00]
39.95: (DELIVER_VICTIM ACC_VICTIM_1 HALIFAX_HOSPITAL) [10.00]
```

5.4 Method C: GIPO - an object-based GUI

5.4.1 Overview of the Method C

The main goal of GIPO is to abstract the syntactic details of the modelling language and helps the user to create a domain model by using graphical metaphor (McCluskey and Kitchin, 1998). The method provides many advantages for modeling and acquisitions domain knowledge.

It is inspired by formal methods for software engineering, and led to the creation of the knowledge engineering platform GIPO (McCluskey and Simpson, 2006; Simpson et al., 2007). Central to this approach is the precise definition of a planning state as an amalgam of objects individual states. This gives us the concept of a *world state* as one being made up of a set of states of objects, satisfying certain types of constraints. Operator schemas are constrained to be consistent with respect to the state, giving the opportunity for using tools to do consistency checking.

5.4.2 Execution of the Method C

In GIPO user need to follow steps to construct domain models. The steps are discussed in following while constructing the RTA domain model for OCL_h:

1. **Sorts and Objects:** Identify a set of *dynamic object classes*, and a class hierarchy (Figure 5.11), where objects inherit state and behaviour from classes in the hierarchy above it; thus an object might have superclass mobile and inherit a range of possible states (describing positions) from the mobile class which the mobile object might occupy (hence defining the range of behaviour). The object might also be a carrier, and inherit state and behaviour from this also.

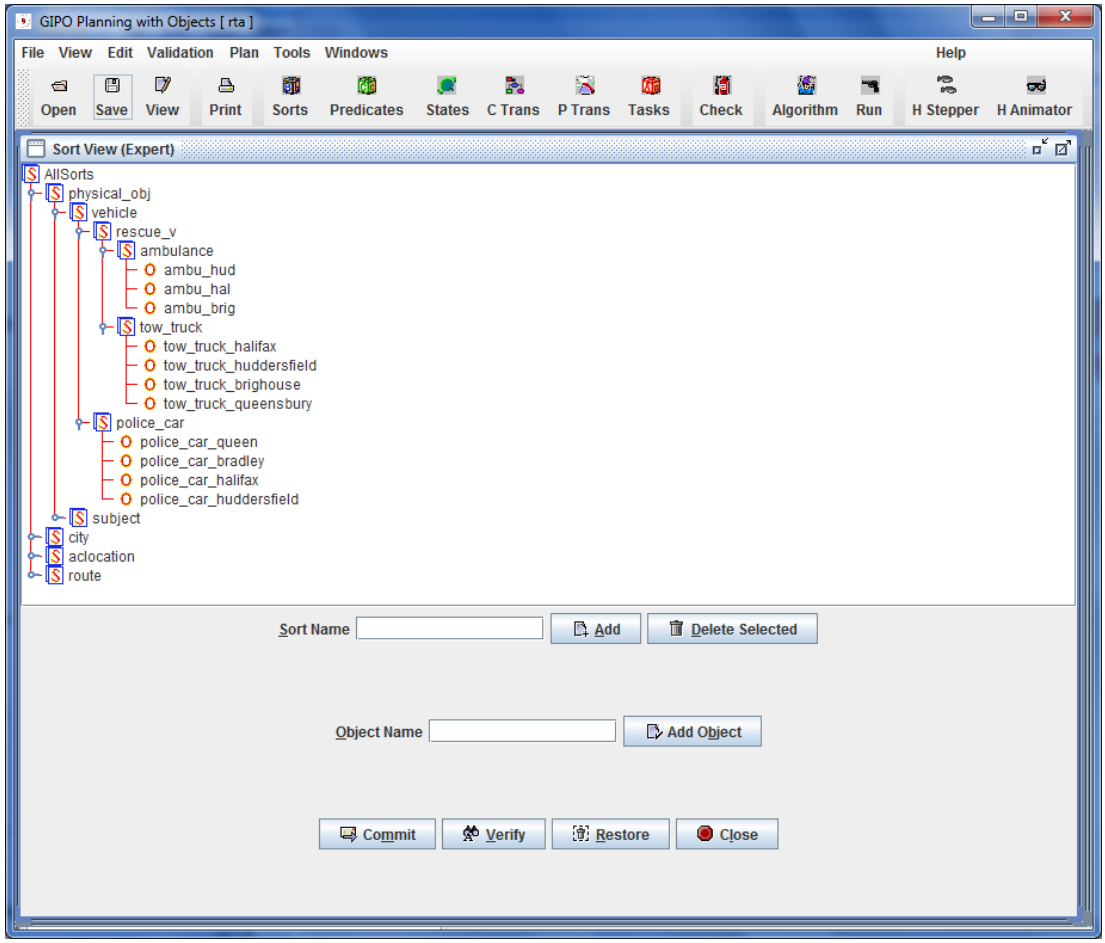

Figure 5.11: GIPO-III Sort (Class) Hierarchy

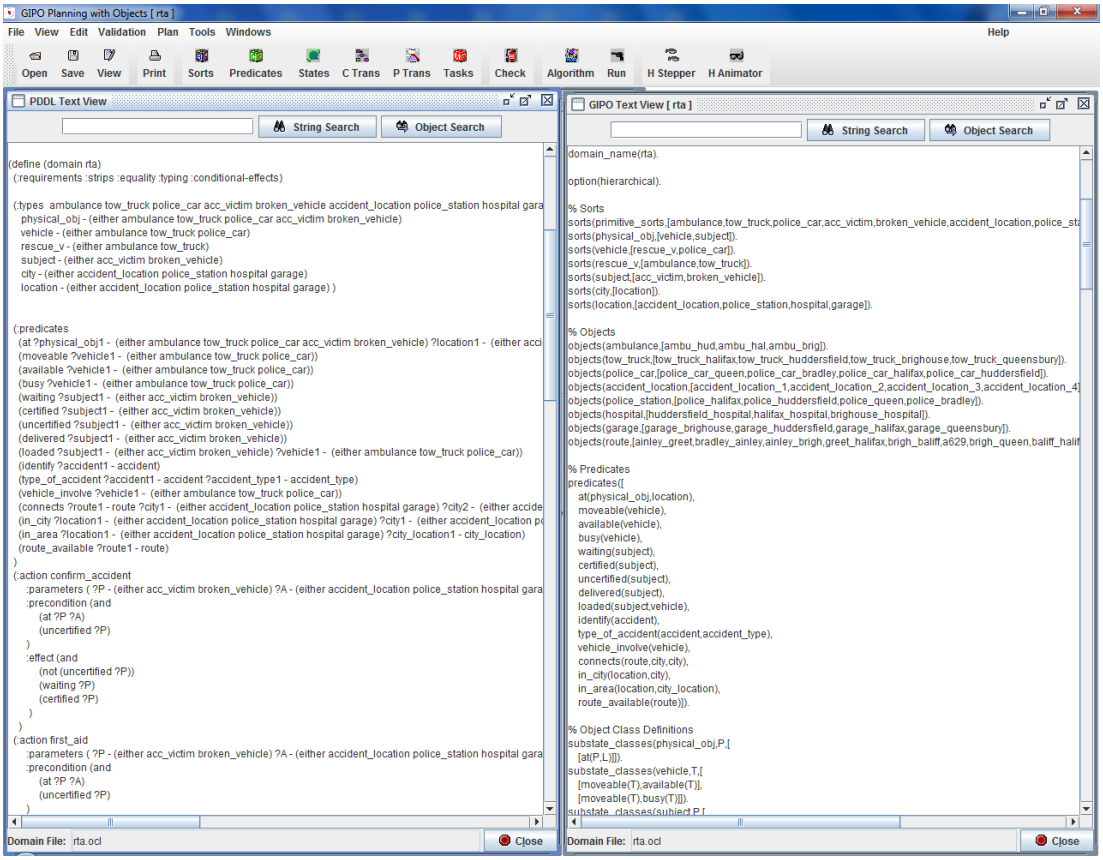

Figure 5.12: GIPO-III Produced PDDL and OCL domain model

GIPO uses sort editor to create sorts rather than doing direct encoding. It is also possible to create many objects by using the dialog boxes in the sort editor window as shown in the figure 5.11. The editor also allows the user to verify the objects and sorts. GIPO generates the domain code in *OCL* and PDDL1.2 which can be viewed by clicking the view option. Figure 5.12 shows the PDDL and OCL (see Appendix E for full code) codes for Road Traffic Accident (RTA) domain.

2. **Predicates:** In the OCL and PDDL domain model, a set of predicates are used to express the relationship between the sorts. Predicates are defined after the sorts

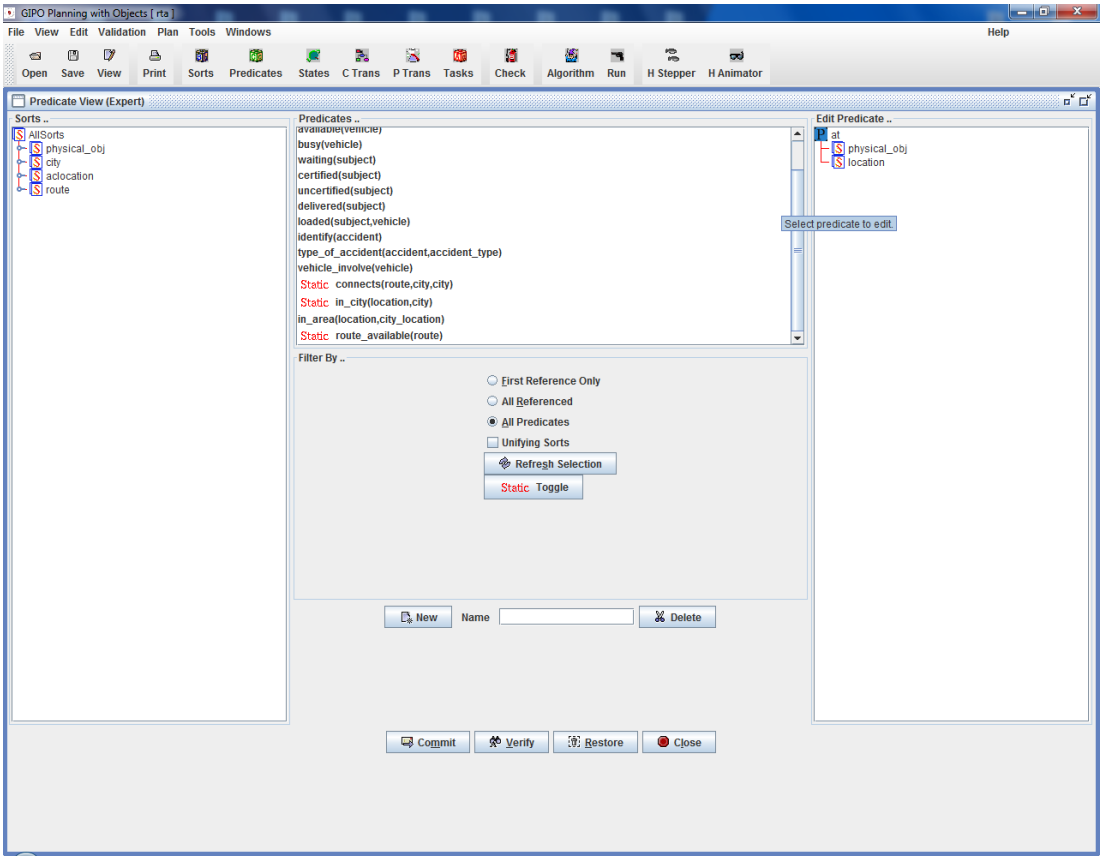

Figure 5.13: GIPO-III: Predicate Construction

have been defined to express various states of objects in the domain model. Once the predicates are created by using GIPO they keep the same arity for further validation. Some of the predicates have single arity, such as moveable (vehicle) and available (vehicle) for the vehicle. Some of the predicates have more than one arity such as connects(route,city,city) and loaded (subject,vehicle). Also, the predicates in GIPO can be declared as static or dynamic. A static predicate like *connects(route,city,city)* which does not change during the execution of planning algorithm. Figure 5.13 shows the process of creating predicates from the sort defined earlier.

3. **States:** GIPO state definition formally specifies constraints on the state of possible objects, i.e., the state of any object in any certain phase. For example, a vehicle 'ambulance' should be available and movable before it can be used or movable and busy when it is in used. It is not possible for a vehicle to be busy and available at the same time as shown in the following -

```
substate_classes(vehicle,T,[
  [moveable(T),available(T)],
  [moveable(T),busy(T)]]).
```

4. **Invariants:** Invariants are defined in the *OCL* language to make explicit assumption about the domain model. Atomic Invariants are used in OCL_h and can be modelled in GIPO tool to set the compatibility boundary of objects. They often contain a list of predicates to specify some instances. For example, in the RTA domain, the routes connect cities have unique names and cannot be changed. They are modelled in the atomic instruments. The following OCL_h code shows part of the atomic invariants created by GIPO -

```
% Atomic Invariants
  atomic_invariants([
    in_city(accident_location_1,ainley_top),
    in_city(police_huddersfield,huddersfield),
    .......
    .......
    route_available(ainley_greet),
    connects(ainley_greet,ainley_top,greetland),
    ........
```

.....

connects(ainley_halifax,halifax,ainley_top)]).

5. **Operators:** Hand-coding operators (for both PDDL and OCL_h) are the hardest part of the domain model construction that requires careful planning and time. To write the code of an operator a knowledge engineer must have expert knowledge on the language syntax and semantics. A knowledge engineer builds a set of operators by using GIPO's transition constructor which does not require any expertise on OCL language. The figure 5.15 shows the GIPO's transition editor. There are five windows in the transition window as shown in the figure. The substate list of any sort can be viewed by selecting the relevant sort. The Editing/Drawing Canvas helps to create the operator by drawing some diagrams as shown in the figure 5.15. The tool helps the modeller to identify the static predicates that cannot be changed by the operation of an operator. The user will be able to use the states of any particular object and will not be able to add anything extra at this point if they were not defined previously. In case of any ambiguity, GIPO warns with pop-up dialog box.

6. **Methods:** Develop HTN *methods* in a top down manner, reflecting the main requirements of the domain tasks, followed by development of primitive operators.

7. **Task:** The HTN task in OCL_h is like the planning problem that contains the initial and goal states.

8. Use static analysis, implemented through tools, to help remove defects from the model as it is developed. Examples are to check operators do not invalidate state

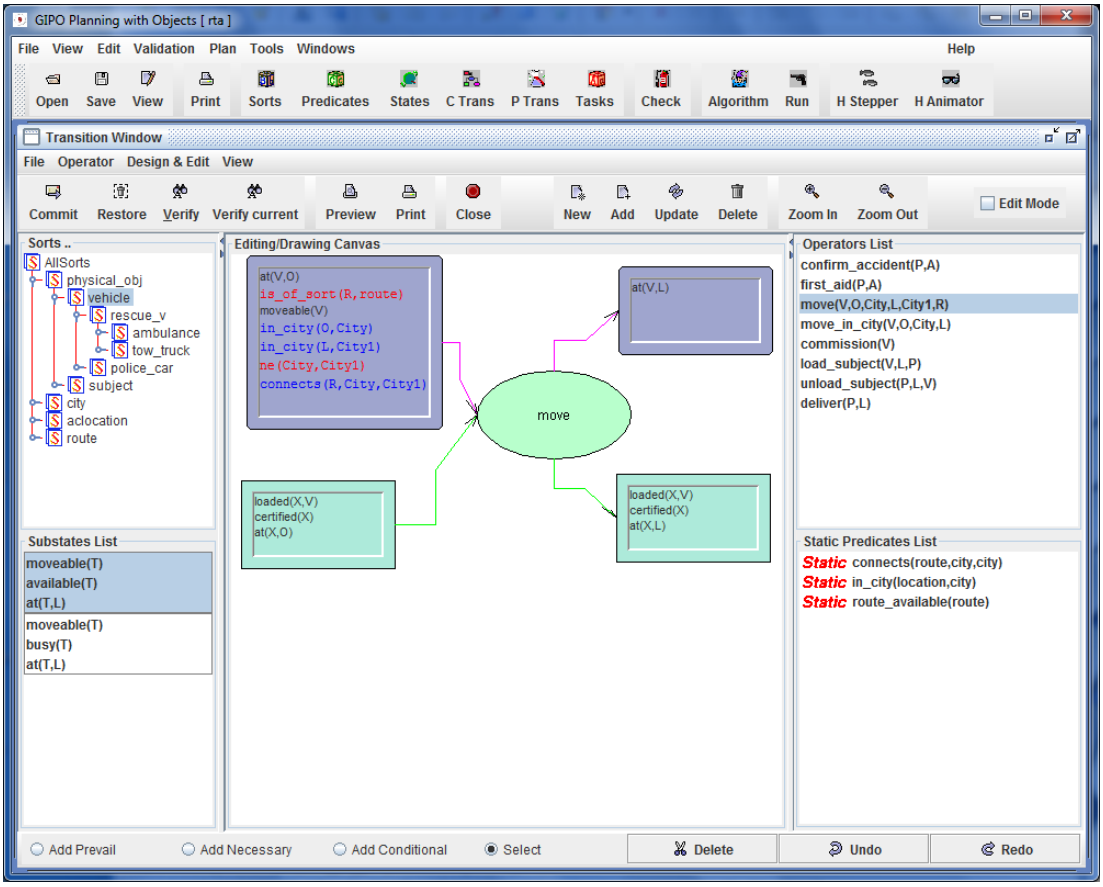

Figure 5.14: GIPO-III: Transition Editor to Create Operator

constraints, and tasks obey the *transparency property* (McCluskey and Kitchin, 1998).

The main modelling language of GIPO is OCL_h that influences the design process of the domain model. The model was encoded with the help of tools in GIPO-III, using basic consistency checkers as well as the more complex transparency property checker. Hierarchies of classes were used to capture state: for example, in the RTA domain, in a particular state an ambulance may have a position (as it is a physical asset), be in service and available (as a movable asset). The set of constraints was added using GIPO-III to encode each of the dynamic object class's possible behaviour that is the

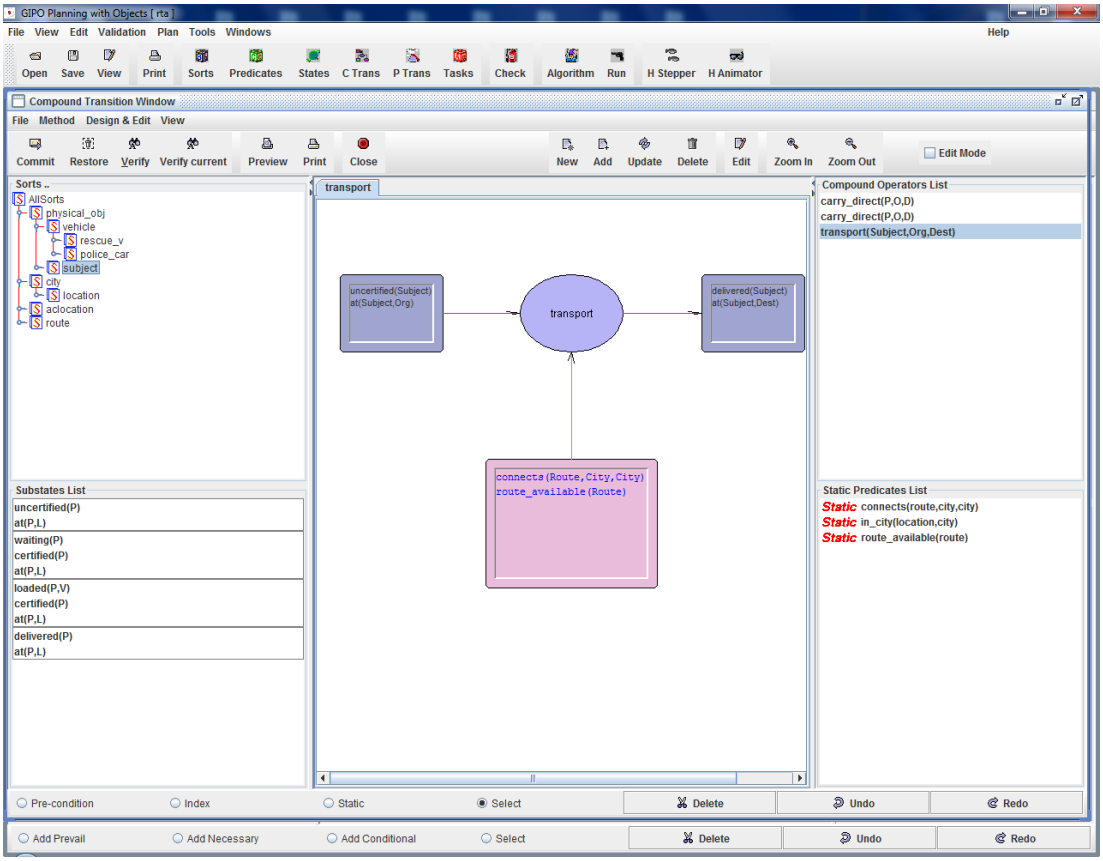

Figure 5.15: GIPO-III: Creating Compound Operator (Method) using Compound transition Window

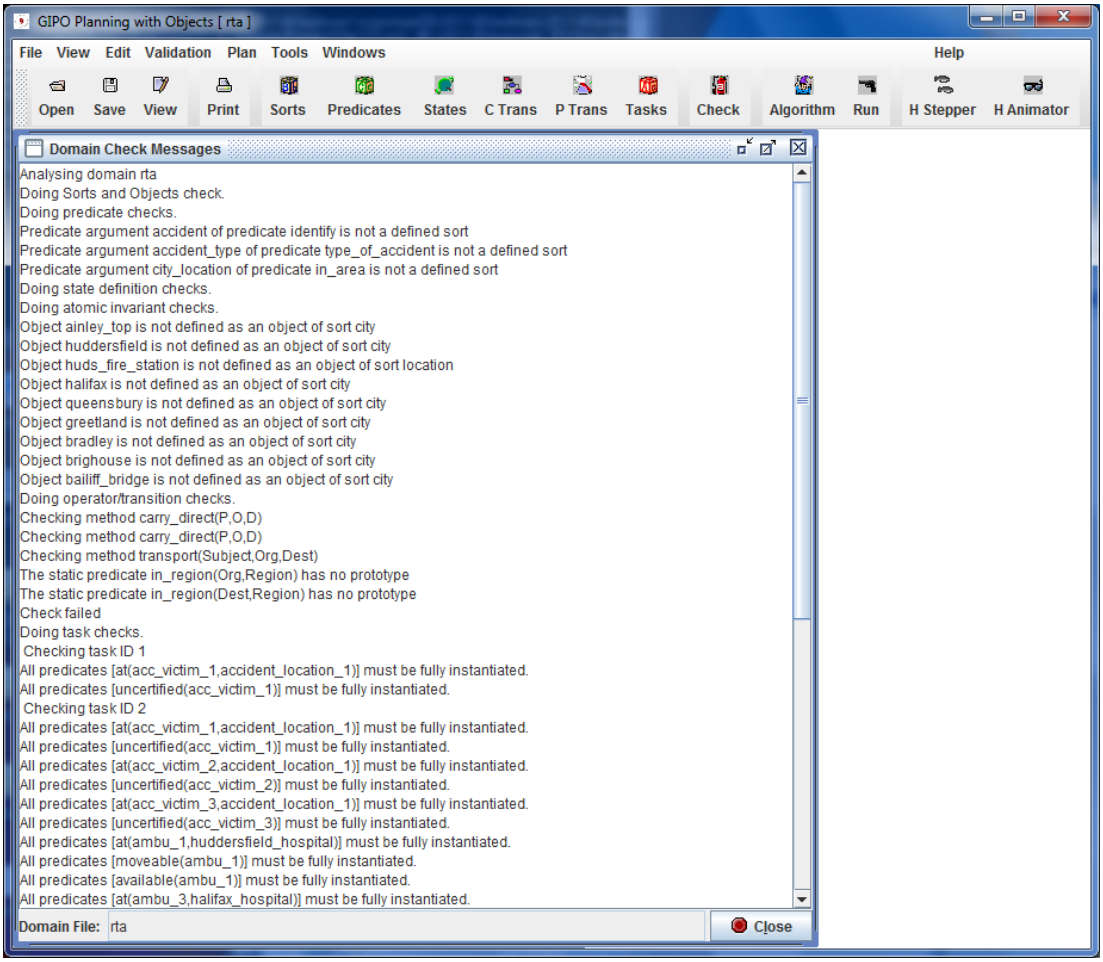

Figure 5.16: GIPO-III: Domain Consistency Check Report

range of states that each object can occupy.

An *action* in an OCL_h domain model is represented by either a primitive operator or a compound task. Primitive operators specify under what condition objects may go through single transitions; compound actions specify under what condition objects go through a sequence of ordered actions. 'Achieve goals' are used to implement pre-condition planning within the HTN framework (effectively one can encode both HTN and goal achievement planning within OCL_h).

In the final step, the model was run repeatedly through GIPO-III's tool set, which checks operator consistency, and transparency properties of compound tasks. Initially the test result shows all the errors, for example, (see figure 5.16) a place called 'ainley_top' was used without declaring earlier so, the GIPO checks the operator consistency and produce a report. The tool was run approximately 5 times on the model, with debugging taking place using the information from each run. After the model passed all the tests, the model was run with the *HyHTN* (McCluskey et al., 2003) planner for plan generation. Basic acceptability test for plan generation is provided in the following subsection.

5.4.2.1 Method C: Basic Acceptability Test

Test Case OCL_h Domain Model: In this test we set the goal to take ACC_VICTIM_1 (Accident victim) to hospital.

Test Result: After a number of trials, we found the solution from *HyHTN* planner. We check the solution carefully to identify any unusual behaviour. The test is passed which implies that the domain model and the problem instance are acceptable.

```
move(police car bradley,police bradley,bradley,acc loc 1,ainley top,
                                                    bradley ainley)
confirm-accident(police car bradley,acc victim 1,acc loc 1)
commission(ambu hud)
move(ambu hud,huddersfield hospital,huddersfield,acc loc 1,ainley top,a629)
first-aid(acc victim 1)
load-victim(ambu hud,acc loc 1,acc victim 1)
move(ambu hud,acc loc 1,ainley top,huddersfield hospital,huddersfield,a629)
unload-subject(acc victim 1,huddersfield hospital,ambu hud)
```

```
deliver(acc victim 1,huddersfield hospital)
```

5.5 Summary

This chapter presented the essential parts of designing and developing process of Road Traffic Accident (RTA) domain by using three KE methods. It is important to be aware of that this chapter did not cover all the aspects of the modelling of the domain by using those methods. For example, we have not covered how itSIMPLE translate the UML to PDDL or how GIPO checks operator consistency. On the other hand, hand-coding for any domain depends on the expertise of the modeller about the PDDL language syntax and semantics. Nevertheless the contents of this chapter should be adequate for understanding for the issue raised in this thesis and the readers are encouraged to itSIMPLE and GIPO manuals and publications. The essential point here is that the knowledge engineering experience by using different methods. The experience of developing the same domain model (RTA) by using different methods gives insights about the knowledge engineering tools and techniques. It has been noted that developing different models do not take much different time but the experience of using different methods are distinct. The evaluation and comparison of the three methods will be discussed in the following chapter.

Chapter 6

Evaluation of Knowledge Engineering Tools and Techniques

Having analysed the Road Traffic Accident (RTA) application, and conceptualised the requirements, a number of domain models have been developed using three different knowledge engineering techniques. The domain models are developed by knowledge engineers where their skills and methods may significantly influence the quality of the product (domain model). The quality of domain model may influence the performance of the planners, expertise of knowledge engineers or the quality of modelling techniques. On the other hand, there are few guidelines or standard procedures available for knowledge engineers to develop a planning domain model.

This chapter evaluates the knowledge engineering tools and techniques for formulating domain models using three different approaches:

(i) observation of the process(development) and product(domain model),

(ii) using set criteria

(iii) using ICKEPS evaluation criteria

All the evaluations have been performed using the road traffic accident (RTA) domain model, described in Chapter 4 and developed in Chapter 5 with an experimental scenario.

6.1 Observation of the methods

The aim of this experiment is to evaluate the three approaches to formulate a knowledge model, with the novel RTA domain. 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 source documents and expertise, until it reaches a final form in which it can be input to planner(s). Features such as defect removal and the nature of the testing process are considered here.
- the **product** of the formulation: this is the domain model and problem files, with measured features such as performance, quality of plans produced and range of problems solved (where they are consequences of the model rather than the planner). Other features include size and complexity.

6.1.1 Actors Involved in Domain Development

We set up an experiment to evaluate three 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). Developing domain models using three

methods involved six actors in total. 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 experts involved in the requirement phase. The background of all participating in the teams was that all except two (the author of this thesis and a colleague) had a PhD in AI Planning. The expertise level of the PDDL encoders (method A) was expert where as for method B and C the team leaders were competent rather that expert in the use of the itSIMPLE and GIPO respectably.

6.1.2 Experimental Scenario

For evaluating the methods we used a set of test instances, considering the map shown in Figure 6.1, 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.

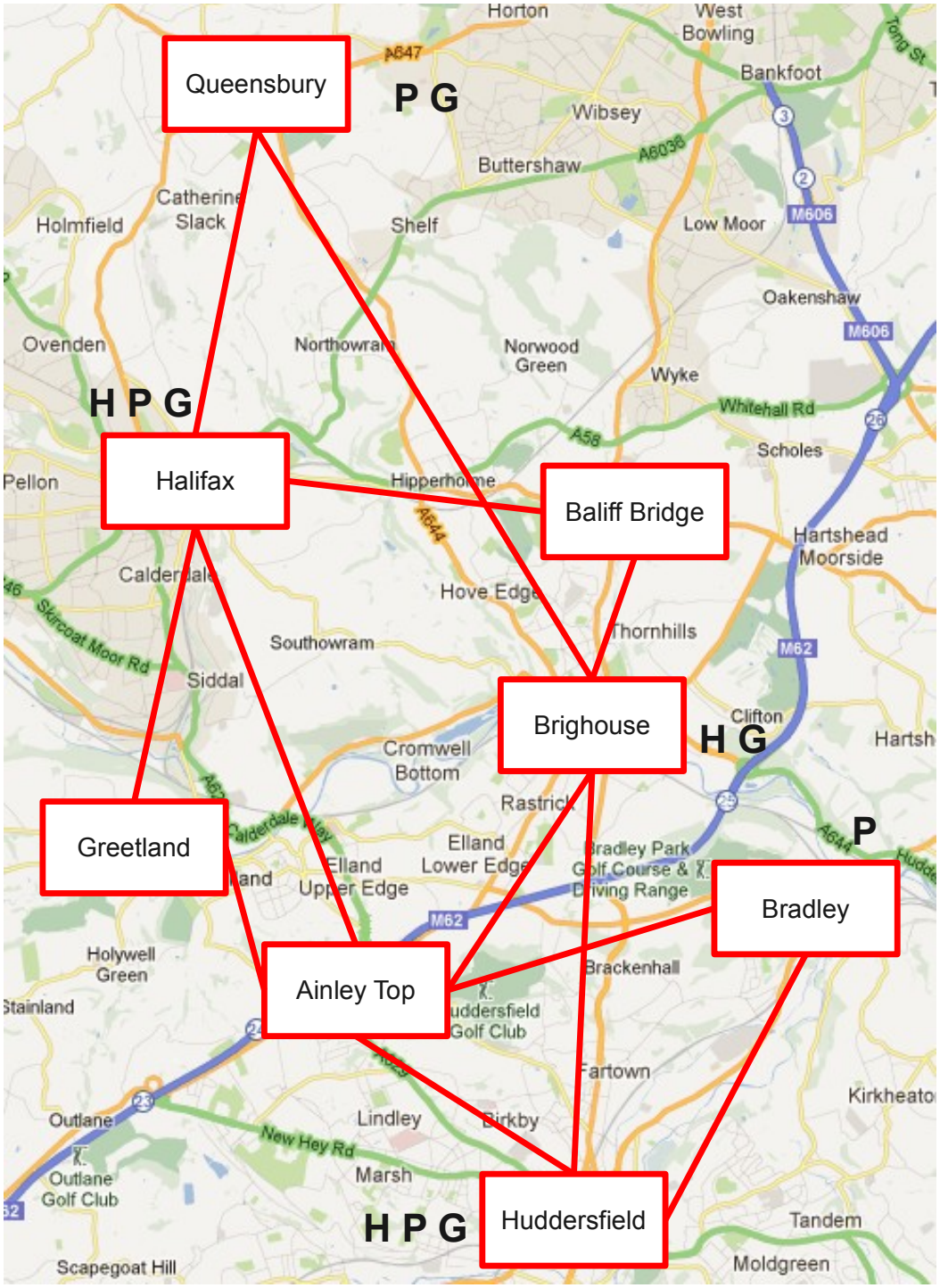

Figure 6.1: The Road Traffic Domain Model used for empirical analysis. It consists of a portion of the county of Yorkshire. H, P and G respectively stand for Hospital, Police station and Garage locations.

6.1.3 Observations of Method A

6.1.3.1 Process

The model took approximately two days to complete using one expert and one competent person using a text editor. It should be noted that most of the time required by this method has been spent in dynamic testing: analyzing produced plans, identifying bugs and removing them from the model. Usually many issues are noticed only by carefully reading the generated plans with the hand encoding domain model. One example of bug identification is when we noticed broken vehicles were being delivered in hospitals, instead of garages. Removing the bug in this case amounted to added further constraints to the operators.

The main issue related to 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 judgment of the expert that is working on it. This lack of structure leads to domains that, even if representing the same real world application, are very different and usually very hard to understand if developed by different experts. Moreover, while everything is left to the sensitivity and to the knowledge of an expert, the process of this method is hard to replicate.

There are advantages in using hand-coding over using well developed tools supported environments: the latter tend to lag behind in the use of expressive modelling languages. For instance, OCL_h does not allow numerical constraints to perform arithmetic operations, such as evaluating distances and the time required for movements, nor *durative actions* in order to facilitate temporal constraints in planning. Even itSIMPLE, which is being continually developed, has some limitations in the type of PDDL that it can generate.

```
30:   (LOAD_VICTIM AMBU_HUD GREETLAND VICTIM_6) [5]
35:   (UNLOAD_VICTIM VICTIM_6 HALIFAX_H AMBU_HUD) [5]
47:   (LOAD_VICTIM AMBU_HUD BALIFF VICTIM_5) [5]
52:   (UNLOAD_VICTIM VICTIM_5 BRIGHOUSE_H AMBU_HUD) [5]
57:   (LOAD_VICTIM AMBU_HUD BALIFF VICTIM_4) [5]
71:   (UNLOAD_VICTIM VICTIM_4 BRIGHOUSE_H AMBU_HUD) [5]
84:   (LOAD_VICTIM AMBU_HUD AINLEY VICTIM_3) [5]
89:   (UNLOAD_VICTIM VICTIM_3 BRIGHOUSE_H AMBU_HUD) [5]
102:  (LOAD_VICTIM AMBU_HUD AINLEY VICTIM_2) [5]
107:  (UNLOAD_VICTIM VICTIM_2 BRIGHOUSE_H AMBU_HUD) [5]
119:  (LOAD_VICTIM AMBU_HUD AINLEY VICTIM_1) [5]
124:  (UNLOAD_VICTIM VICTIM_1 BRIGHOUSE_H AMBU_HUD) [5]
```

Figure 6.2: An overview, showing only *LOAD* and *UNLOAD* actions, of the LPG Planner solution to the basic acceptability test instance. The three columns represent the time at which the action occurs, the action performed and the duration of the action. Brighouse_H and Halifax_H stand respectively for the Hospital in Brighouse and the Hospital in Halifax.

6.1.3.2 Product

We found that the iterative process of debugging described above led to a domain encoding over-constrained, in order to avoid unwanted behaviours. Many of the constraints, added under the form of pre- and/or post- conditions, were built up incrementally during debugging in an ad-hoc fashion. The resulting model is complex and hard to read and understand. The length of the domain file: 225 lines for the PDDL and 248 for the PDDL2.1 model. We are aware that the number of lines is not a very informative indicator, but it gives intuitively a quantitative idea of the size of a model.

In plans generated exploiting method A, usually a very small number of vehicles are employed, resulting in high plan duration. Figure 6.2 shows an overview of the plan generated by the LPG Planner for the basic acceptability test instance; a single ambulance (*ambu_hud*) is used for transporting all the accident victims to different hospitals, while other ambulances are not considered (and we found the use of tow trucks to have the same behaviour). This behaviour arises due to the high number of pre-condition and effect of operators in method A domain models prevented the

planner using concurrently different emergency vehicles. Mean number of positive pre-condition and effects in this method are 4.3 and 1.7 respectively for both PDDL and PDDL2.1 domain models.

6.1.4 Observations of Method B

6.1.4.1 Process

The model took approximately three days to build and debug using one person competent and one in support in using the tool. Most of the time was spent in designing classes of objects and defining legal interactions between them. After that, only a relatively short time is required for debugging. Every step was effectively supported by the KE tool used.

Since the itSIMPLE tool has been designed for supporting a disciplined design cycle and for supporting the transition of requirements to formal specifications, the process is clearly defined and easy to repeat. A user goes through the formulation of requirements by designing several UML diagrams, and automatically generates the corresponding PDDL (or PDDL2.1) domain encoding. The tool gives the modeller a range of third party planners for generating plans, along with features such as 'plan analysis'. However, we found that most debugging is 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 (in contrast to method C).

6.1.4.2 Product

We observed that the structured and principled process of encoding the requirements led to domain encodings that are clear and easy to understand. Evidence to support this is given by the length of domain description: 263 lines for the PDDL and 259 for PDDL2.1.; about 10% more than the hand written models resulting from method A. Also, mean positive pre-condition and effects in this method are 3.8 and 1.4 for PDDL, and 4.0 and 1.6 for PDDL2.1 domain model provided in the table 6.1. On the other hand, the UML documentation is useful in maintenance, as it helps trace the encoding to the initial requirements. The good quality of the encoded domain models leads to better quality of plans.

6.1.5 Observations of Method C

6.1.5.1 Process

The OCL_h model was debugged using the tools within GIPO. The first run identified syntax and typographical errors such as differences in predicate names, which were corrected. During the second run, while doing operator/transition checks, inconsistencies in the HTN methods were revealed which led to another round of bug removal. Part of the output of the consistency checking tools, for one early run, is shown in Figure 6.3. In addition to checking the modelling of the domain, GIPO can also check that tasks are defined correctly and in a way which is consistent with the domain model. The second run gave only instantiation warnings for some of the tasks. In addition to debugging with GIPO, dynamic testing via plan generation can also be carried out.

The creation of a dynamic hierarchy of object classes encoded constraints of the domain explicitly, and this is what is used by GIPO's tool support to check operator

```
...
Checking method carry_direct(P,O,D)
found an unrecognised decomposition item: unload_subject(P,D,V)'.
Check failed
Checking method carry_direct(P,O,D)
found an unrecognised decomposition item: unload_subject(P,D,V)'.
Check failed
Checking method transport(Subject,Org,Dest)
The static predicate in_region(Org,Region) has no prototype
The static predicate in_region(Dest,Region) has no prototype
Check failed
Doing task checks.
....
```

Figure 6.3: Part of output from GIPO: here the transparency of HTN methods is checked and found to fail, with the likely faulty components identified.

schema, states, predicates etc., and identify bugs prior to dynamic testing. For example, the range of behaviour of a *movable asset* possess such as an ambulance is specified by state constraints, before operators are constructed. Hence, important constraints of the domain, rather than being implicit as in PDDL, are explicit in the model's specification. As observed above, initial tests with the PDDL hand crafted model allowed ambulances and tow trucks to have strange behaviours such as carrying cars to hospitals (rather than garages). This behaviour was eliminated during domain specification in Method A. While one could argue that dynamic testing picked this up anyway, there may be behaviours that we have not picked up in the tests up to now that result from hidden bugs.

The model took approximately four days to build and debug using one competent person as the lead and one person in support. The method took significantly more resource than the others, because of the unfamiliar notation compared to UML or PDDL, and the encoding of complex HTN methods.

move(police_car_bradley,police_bradley,bradley,acc_loc_1,ainley_top,bradley_ainley)
confirm-accident(police_car_bradley,acc_victim_1,acc_loc_1)
commission(ambu_hud)
move(ambu_hud,huddersfield_hospital,huddersfield,acc_loc_1,ainley_top,a629)
first-aid(acc_victim_1)
load-victim(ambu_hud,acc_loc_1,acc_victim_1)
move(ambu_hud,acc_loc_1,ainley_top,huddersfield_hospital,huddersfield,a629)
unload-subject(acc_victim_1,huddersfield_hospital,ambu_hud)
deliver(acc_victim_1,huddersfield_hospital)
...

Figure 6.4: Part of the HTN solution plan for the basic acceptability instance. The sequence allows to deliver an accident victim to the hospital in Huddersfield.

6.1.5.2 Product

The above process led to a model which when used with GIPO's HTN planner HyHTN found a plan which was in accordance with current standards of how planning is being done by Rescue Services. The size of the OCL_h model is larger than the size of the PDDL models, because state constraints are encoded explicitly, and HTN methods are specified in addition to primitive ones; the OCL_h model has 496 lines of code, it is about twice long as the PDDL models generated by other methods. Structures of the solutions, a part of the solution generated by the HyHTN planner is shown in Figure 6.4, are similar to solutions generated by the LPG planner on the PDDL model developed in itSIMPLE (the method B). A possible advantage of this approach we believe might be better scalability than the PDDL models.

6.2 Evaluation of the methods from Observation

6.2.1 Process Comparison

Regarding method A, the main issue related to 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 judgment of the experts that are working on it. This lack of structure leads to domains 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 has been 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 was 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. Moreover a hand encoding domain model contains many issues are noticed only by carefully reading the generated plans. One example of bug identification is when we noticed broken vehicles were being delivered at hospitals, instead of garages. Removing the bug in this case amounted to added 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 the Method C tool GIPO (Simpson et al., 2007) which is capable of identifying bugs at an earlier stage than dynamic testing. Regarding the construction of the RTA domain model in the HTN language (OCL_h) utilising GIPO, we observed that the extensive use of static tools resulted in a model that required less dynamic testing during the debugging phase. An advantage of using an HTN encoding, however, is that methods compound 'building blocks' of plans and therefore allow developers to encode intuitive plan traces. 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 a 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, 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 dialogue boxes within GUI-based tools, which hamper their ease of use.

One final point: once a PDDL model has been generated, if it is maintained outside of the tools environment, it is in general not possible to reverse engineer it back into the environment. This means that the environment tends to be used only for initial encoding, and not over a domain model's life cycle.

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

6.2.2 Product Comparison

For comparing the domain models generated by methods A and B, we selected a subset of the metrics suggested by Roberts et al. in (Roberts et al., 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 models 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 domains. The results of this comparison between Method A and B are shown in Table 6.1, 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.

We found that the iterative process of modelling and testing domain model in Method A led to an over-constrained domain encoding. Many of the constraints, added under 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. 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 the 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 generated by method B has 2 more actions than method A one; these operators are related to the untrapping people and extinguishing fire tasks and are used for avoiding that the same fire brigade extinguishes several fires and untraps several people at the same time. The method A domain model exploits a "trick": the PDDL experts added the same predicate as positive and negative effects 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 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 that PDDL plans, since actions are instantly completed, are a very compact version of the PDDL2.1 ones. The simplest 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 simplest 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

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 6.2: 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.

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. The results of the experiment are shown in Table 6.2 in terms of CPU time, number of actions and plans duration.

These results indicate that LPG with the hand written domains needs more CPU time, both in PDDL and PDDL2.1, than with the domains 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 domains are 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 generated by method B slow down the plan generation process, but they 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 its plans having a high makespan.

6.3 Criteria for evaluating approaches

In Chapter 3, we have elicited a set of characteristics after discussing a range of knowledge engineering tools and techniques. Also, we have developed a real application domain model called Road Traffic Accident (RTA) in Chapter 5 using three different knowledge engineering tools and techniques. We evaluate those domain models by using an experimental setup in the previous section. The whole experience directs to some criteria that are useful for evaluating the different approaches for encoding domain models. In this section; firstly we define those criteria, secondly we evaluate each of the methods based on those criteria and finally we provide some important features that the future knowledge engineering tools for planning should possess.

Operationality. How efficient are the models produced? Is the method able to improve the performances of planners on generated models and problems?

Collaboration. Does the method/tool help in team efforts? Is the method/tool suitable for being exploited in team or is it focused on supporting the work of a single user?

Maintenance. How easy it is to come back and change a model? Is there any type of documentation that is automatically generated? Does the tool induce users to produce documentation?

Experience. Is the method/tool indicated for inexperienced planning users? Do users need to have a good knowledge of PDDL? Is it able to support users and to hide low level details?

Efficiency. How quickly are acceptable models produced? How easy is it to use the tool?

Debugging. Does the method/tool support debugging? Does it cut down the time needed to debug? Is there any mechanism for promoting the overall quality of the model?

Support. Are there manuals available for using the method/tools? Is it easier to receive support? Is there an active community using the tool?

6.4 Evaluation of the approaches with respect to stated criteria

This section employs and hence evaluates three separate methods for knowledge formulation: (A) the traditional method of hand-coding, using a text editor and relying on

dynamic testing for debugging (B) itSIMPLE, and (C) GIPO. In the following we will evaluate each method with respect to the criteria stated at the beginning of this Section.

6.4.1 Method A

This method involves developing planning domain in PDDL using a text editor (in this thesis, Gedit), given the description of the real world domain in the Chapter 4.

Operationality. Operationality mainly deals with the performance and the model efficiency. Even if method A (hand-coding) is the most exploited methods for generating new planning domain models, there are no evidences that it leads to models that are more efficient than the ones generated by other methods. The quality of models depends on the expertise of the person that encodes it and is hard to predict a-priori. In fact, we have experimentally observed this method's operationality as follows,

- Performance - Since the quality of the product developed in this method depends on the expertise of the knowledge engineer, in which it is difficult for an expert to cope all the expressiveness of the language.
- Model Efficiency - The models generated by this method reduce the performances of planning algorithms if the model is not developed carefully.

Collaboration. This method does not support any type of collaboration. Usually the model is produced by a single expert, who eventually discusses issues or improvements with domain experts rather than with other planning experts.

Maintenance. It is usually easy, for the expert that encoded the domain, to come back and change the produced model. On the other hand, usually models are not

documented. This means that the maintenance is potentially hard, in terms of the complexity of the model, for people that were not involved in the encoding process. Some of the features related to the maintenance can be summarised as below -

- Reverse engineering - This method does not require to develop any graphical metaphor, so there is no need for reverse engineering. One of the advantages of using text editors is that it can be used on any operating system platform.
- Change handling - It is sometimes hard to adapt to change in one part with the whole domain model. For example, if we require changing object type, it will require changing all the respective parts of the domain model. In the hand-coding method it is difficult, since the user has to find and change manually. This manual change can lead to the wrong output plan as a result of wrong encoding.

Experience. This method is indicated only for PDDL experts. Experts know the ways for handling some common issues and are able to interpret the planners output in order to identify bugs. From the experience of developing an RTA domain model by hand, it is clear that encoding a new domain model from scratch is very hard and error-prone process. Moreover, the process of dynamic testing is time consuming which clearly indicates that this method does not provide any positive user experience.

Efficiency. Usually a first version of the model is quickly produced. This leads users to perceive this method as a very efficient one. On the other hand, the first version

requires a lot of dynamic tests to become acceptable. The design efficiency of this method can be summarised as below -

- Requirement Analysis - the requirement analysis entirely depends on the user. There is no guideline provided how to encode domain models in this method.
- Design Process - the user may or may not follow any design process. From the experience of designing the RTA domain model, we can say that it is worth to follow a process to reduce development time.
- GUI Support - this method is straightforward method, which does not require to draw a diagram or model, to encode the domain model by using any modelling language.
- Design Speed and Acceptability - since there is no diagram to draw, it seems that the modelling process takes less time. In contrast, we can see that designing the larger real-world domain model requires a great number of operators, predicates and objects, which significantly slows down the process as it is hard for a single user to keep track all the elements for a single user.

Debugging. Debugging while hand-coding a model is a critical task. The only way for debugging is dynamic testing. This involves the use of one or more planners for solving some problem instances, then an analysis of the produced plans for identifying bugs, and finally remove them from the model and start again. Removing bugs are usually done by adding further constraints to the operators. In the following we discuss the debugging during the domain development using method A.

- Syntax Error - We have used Gedit for developing domain model. There is no facility to check the syntax of PDDL in this text editor.
- Semantic Error - Text editors do not help knowledge engineers to identify semantic of domain developing language like PDDL.
- Static Validation - Text editors do not support any static validation such as validation of objects of types, valid predicate etc.
- Dynamic Validation - The hand-encoded domain model must rely on the dynamic testing by using an appropriate planner. The process is iterative as each of the iterations we have to remove the error and run the planner again. The process continues until a valid plan is found for a given planning problem.

Support. There is no guidelines available to model real-world problem using this method. So, developing domain models left to the planning experts.

6.4.2 Method B

This method involves a user that exploits itSIMPLE for generating new planning domain models. The steps of the method described in the Chapter 5. In this section, we discuss the selected criteria for KE tools and techniques using the experience of building the RTA domain model by using itSIMPLE.

Operationality. Our experimental analysis indicates that usually domain models generated by itSIMPLE improve the performances of planners. This is probably due to a less constrained domain description that derives from the UML diagrams.

- Performance - The quality of the product developed in method B depends on the knowledge engineer's ability to use the tool. Also model developed using itSIMPLE took less CPU time during plan generation.
- Model Efficiency - Although, the numbers of lines are around 10% more than the hand-coding domain model but the produced model is more efficient as it is clearer to read and maintain.

Collaboration. itSIMPLE has been designed for a single user. However, it is possible to import models (projects) developed by different users. This helps the exchange of ideas and comments among the users.

Maintenance. The itSIMPLE tool is designed for supporting a disciplined design cycle. The UML diagrams can also be used as documentation. From this point of view it is easy to maintain a generated domain model.

- Reverse Engineering - If a model generated by itSIMPLE is changed in different tool or even a text editor, then it is not possible to maintain the model in itSIMPLE environment.
- Change Handling - Changes in the domain modelling must follow the UML rules in order to minimise error.

Experience. Typically users are not required to be PDDL expert, but he should have some basic knowledge of the language as well as UML. There are some important aspects of user experience when developing domain models in itSIMPLE.

- The experience of using the tool is important as an expert user of itSIMPLE will develop better domain model than the novice one.

- Good knowledge of PDDL language will ensure better domain model.
- Using itSIMPLE for creating PDDL problem file is much easier than other methods. Problem files are created by using the objects using UML design.
- itSIMPLE comes with a range of state-of-the-art planners, which allow user to perform dynamic testing during the modelling phase.

Efficiency. Most of the time is spent in designing classes of objects and defining legal interactions between them in UML. After that, only a short time is required for debugging. The total time spent per domain was about a person week.

- Requirement Analysis - itSIMPLE is one of the tools that introduces requirement modelling as software engineering. Designing the UML diagrams help non-AI experts to design PDDL domain models.
- Design Process - itSIMPLE uses a semi-standard design process. One of the important features of this design process is the post design analysis to improve the domain by analysing the plan.
- GUI Support - itSIMPLE provides GUI support for the user. User design use-cases, class and state machine diagram in the GUI environment. The GUI also helps to translate the model into PDDL and generate plan using planners.
- Design Speed and Acceptability - The initial design process is slower in itSIMPLE, i.e., requirement analysis. Once the requirement is designed, creating problem instances and testing becomes relatively faster.

Debugging. Most debugging initiates through dynamic testing: while the UML description of the models helps in development and maintenance, it is the failure

of a planning engine to solve a goal that alerts the user to the presence of a bug, in most cases.

- Syntax Error - itSIMPLE produces domain models by translating from the graphical representation to the domain definition languages, which reduce syntactic error than hand coding method.
- Semantic Error - itSIMPLE tool does not highlight the code like PDDL-Studio but indicates any undeclared predicates and undefined objects after translation into PDDL. It also prevents incorrect association between objects and classes to reduce error in domain models.
- Static Validation - itSIMPLE does not provide any facilities of static validation but the UML diagrams such as, class diagram defines the relationship between classes which must be met to build operator transition in the object definition. So, this embodies a way of static validation.
- Dynamic Validation - itSIMPLE comes with a range of planners integrated to perform dynamic validation of a domain model. It is one of the major advantages of itSIMPLE to allow the range of planners. It is convenient for users to design and test for valid plan whereas the traditional hand coding domain model has to be tested manually.

Support. itSIMPLE provides documentation which includes a description of the tool, a tutorial and a FAQ section. It is easy to find information and solutions for most of the common issues.

6.4.3 Method C

The domain models are encoded with the help of tools in GIPO-III, using basic consistency checkers as well as the more complex transparency property checker. Hierarchies of classes are used to capture state: for example, in the Road Traffic Accident domain, in a particular state an ambulance may have a position, be in service and available. The set of constraints is added using GIPO-III to encode each of the dynamic object class's possible behaviour that is the range of states that each object can occupy.

Operationality. Structures of the solutions are similar to solutions generated by the LPG planner on the PDDL model developed in itSIMPLE. The operationality of the domain model is evaluated with the following features.

- Performance - The performance of a model depends on the expertise of the knowledge engineer for using GIPO and the associated language OCL_h.
- Model Efficiency - GIPO generated domain model is efficient to test with the planner and produce plan similar to the models produced by other methods. The software comes with only one HTN planner called $HyHTN$. So, it is difficult to compare the performance of the domain model with the PDDL one.

Collaboration. GIPO has been designed for a single user. However, it is possible to import models (projects) developed by different users. This helps the exchange of ideas and comments among the users.

Maintenance. Domain model generated by GIPO is generally maintained in the tool environment. Maintenance of GIPO can be evaluated with the following features of KE tools.

- Reverse Engineering - Like itSIMPLE, once the OCL_h domain model has been generated, if it is modified outside the tool environment, it is in general not possible to reverse engineer into the environment.
- Change Handling - The changes in any part of the domain model within the tool environment can be handled by the user manually. The graphical model of GIPO helps modelers to manage change during debugging and testing.

Experience. The typical user is not required to be an OCL expert, but he should have some basic knowledge of the language or the tool. The experience of using the tool can benefit the user in the following way.

- The GUI support of GIPO gives the user better experience of modelling than the hand-coding.
- Code generated by the tool is clear and easy to read.
- GIPO design process guides modelers from the requirement design and planning.
- GIPO allows the user to check the domain model with the integrated planner.

Efficiency. GIPO does not require creating UML diagram like itSIMPLE but requires some transition diagram to create operators. The design efficiency of using GIPO can be summarised as below -

- Requirement Analysis - Like itSIMPLE, GIPO requires to draw diagrams in order to produce OCL_h domain models.

- Design Process - The design process that is used by GIPO is a formal process which associates with the structure of the language. For example, creating a domain model in *OCL* using GIPO, a user must create *sorts* first before creating any predicates. This ensures less bugs in the produced model.
- GUI Support - GIPO integrates a number of GUI tools to help the user to create *OCL* and OCL_h type domain models. It includes Object Life History (OLH) Simpson (2005) diagram that helps user to draw objects and their relationship.
- Design Speed and Acceptability - Using GIPO it is usually faster to create first steps such as sorts, predicates and states, of the domain model. The process becomes slower during the creation of operators and tasks.

Debugging. The creation of a dynamic hierarchy of object classes encoded constraints of the domain explicitly, and this is what is used by GIPO's tool support to check operator schema, states, predicates etc., and identify bugs prior to dynamic testing. We check the debugging facility of GIPO uses the following features -

- Syntax Error - In GIPO, the initial verification identifies the syntax and typographical error e.g., incorrect predicate names.
- Semantic Error - Each step of domain modelling in GIPO provides the facility to verify the current update.
- Static Validation - Static Validation of GIPO is distinct from the other GUI tool. The validation can be used to check each of the steps of modelling phase which reduces overall error.

Criteria	Hand Coding	itSIMPLE	GIPO	Comment
Operationality	Produced model is not so clear to read	Produced model is clear and readable	Produced model is clear and readable	Tool generated domain models are readable and clear. Planning depends on planning algorithm.
Collaboration	NO	NO	NO	KE methods should be collaborative for real-world application.
Maintenance	Easy for the expert who developed but difficult for other	Easy in tool environment	Easy in tool environment	Domain model should be portable i.e., should be maintained in any platform.
Experience	PDDL experts	Users do not need to be PDDL expert	Users do not need to be OCL expert	Tools should target to the non-AI expert users.
Efficiency	Requires more dynamic test	Efficient to design	Efficient to design	Design efficiency depends on the user expertise.
Debugging	Depends on dynamic test	No formal debugging	Static validation at each step	Every KE method should have some mechanism for early debugging.
Support	Only language support available	Supporting documents available	Supporting documents available	Should be updated for each version.

Table 6.3: Evaluating of KE Tools with set criteria

- Dynamic Validation - GIPO comes with integrated planners to check domain models and to generate plans. It is possible to check the plan validity by GUI tools called plan stepper and animation that comes with the tool.

Support. GIPO comes with documentation which includes a user manual, tutorial and OCL language manual. It is easy to find information and solutions for most of the common issues.

6.4.4 Observation of the Evaluation

It has been observed that creating different models does not take very different amounts of time (taking into account the developers' expertise) while exploiting method A and B. Method C usually requires significantly more time resource than the others, because of the unfamiliar notation compared to UML or PDDL, and the encoding of complex HTN methods. However, in method C creation of a dynamic hierarchy of object classes encoded constraints of the domain explicitly, and this is what is used by GIPO's tool support to check operator schema, states, predicates etc., and identify bugs prior to dynamic testing. While one could argue that dynamic testing will pick bugs up anyway, there may be behaviours that have not been picked up in the tests done in method A and B, that result from hidden bugs. On the other hand the UML description of the domain that is required by itSIMPLE helps to prevent many unwanted behaviours. Method A is the most sensitive to bugs, and the quality of the produced model completely depends on the expertise of the user.

Considering the maintenance of the generated models, method B provides the better instruments for changing a model. The UML description provides a sort of documentation that can be exploited for a quick understanding of the domain model and for applying changes. An issue that we noticed while working with itSIMPLE is that it is not possible to import a model that, even if generated by using itSIMPLE, has been slightly modified with a different tool. This forces the user to make several steps in the framework also for very small changes. In method B, the unfamiliar notation of GIPO does not really make it easy, for a non-expert user, to come back and change the model.

Regarding the generated models, there are several interesting aspects to consider.

Models generated by method A are usually very compact in terms of numbers of lines, predicates and types, but they are usually over-constrained in order to avoid unwanted behaviours. The iterative process of analyzing produced plans, identifying bugs and removing them from the model leads to incrementally add constraints in the form of pre- and/or post- conditions. The structured and principled process of encoding the requirements of Method B usually leads to domain encodings that are clear and easy to understand, even if less compact (around 10% longer) than the ones generated by method A. It is worth to mention that the good quality of the encoded domains leads to better quality of generated plans. The quality of models generated by method C is harder to understand due to the different language. The models are significantly larger than PDDL ones (about two times longer in terms of number of lines) and need an HTN planner to solve corresponding problems. We observed that the structures of the solutions are similar to solutions generated by domain-independent planners on models developed by other methods. A possible advantage of this approach we believe might be better scalability than the PDDL models. However, current OCL_h techniques do not support durative actions, which makes it less interesting in domains where time optimization is critical. Summary of the this evaluation has been provided in Table 6.3.

6.5 Evaluation of KE Methods with ICKEPS Criteria

The main goal of ICKEPS competition is to accelerate the progress of Knowledge Engineering for automated planning research (Bartak and McCluskey, 2006). This series of competition has promoted the development of KE tools and share them to the research community in order to create reliable planning system (Bartak et al., 2010). Although, first two competitions focused on general aspects of KE tools but the latest

ICKEPS focused on the particular aspects. The details can be found on the competition websites[1] although a brief discussion has been provided in Chapter 2. In this section we consider the criteria that have been used in the 4^{th} ICKEPS[2] competition in order to evaluate the evaluations that used in the previous sections of this chapter.

6.5.1 Portability

The tools (itSIMPLE and GIPO) have been downloaded from the authors designated web sites. Moreover, both of the tools execute on common Operating Systems like Windows and Linux. Hence, we can say the tools are portable and not considered as evaluation criteria in this this.

6.5.2 Robustness

This is a software engineering issue to find if the tool is sensible to any domain. In our experiment and experience we found that both of them are general tools and suitable for developing various type domain models. It is possible to extend both of the tools to capture different types of domain models such as continuous planning domain model.

6.5.3 Usability

The users of the tools are mainly planning expert who has expertise on the particular domain modelling languages. Generally, it is easy for the expert to use the tool than the novice. In our evaluation we use *efficiency* to find the usability of the tool from our experiment of modelling RTA domain model. Method B (itSIMPLE) forces the user

[1] http://www.icaps-conference.org/index.php/Main/Competitions
[2] http://icaps12.poli.usp.br/icaps12/ickeps

to design UML diagrams to find the domain model, i.e., the user should have some knowledge of UML as well as the output planning language. On the other hand, the Method C (GIPO) also uses a graphical approach like Object Life History (OLH) and transition diagrams.

6.5.4 Spread of use of the tool

Both of the tools that have been used in this thesis are widely used especially the itSIMPLE for formulating PDDL domain models. This has not been chosen in our evaluation as this does not directly relates to the developing domain model or performance of the tool.

6.5.5 Perceived added value

Both of the tools have great impact on the KE research area. For example, GIPO is one of the first general purpose tools that won the first ICKEPS which inspired the research community to build KE tools. On the other hand, itSIMPLE uses the traditional software engineering method by using s UML design which is helpful for the non-expert user to build domain models without knowing the modelling language.

6.5.6 Flexibility

Both GIPO and itSIMPLE provides guidelines to follow for creating domain models. This criterion is mainly for the competition purpose but from our experience and evaluation we can say that any competent user can create domain models by using those tools.

In the ICKEPS competition the criteria that have been used are mostly irrelevant

to our experiment. In this thesis we mainly consider the usefulness of the KE tool for creating real-world domain model. We also consider the product (domain model) created by the tool whether it is useful, better than compare to the hand-encoding domain models, increase or decrease the plan quality, maintenance, debugging. It has been noticed that the capability of KE tools has greater impact on the domain models during plan generation. For instance, tools that use debugging facility in the early stage of domain development always reduces time during the dynamic testing. Although, there is not ideal tool available for domain modelling but some of the important capabilities could make the domain independent tools richer.

6.6 Insights into the Evaluation Processes

To the best of our knowledge this is the first effort to evaluate the KE tools and techniques for automated planning. It can be noted that the evaluation processes are relatively new in this area of research. In this section, we discuss the evaluation processes that used in this thesis in contrast to the process used in the ICKEPS competition. Also, we highlight the advantages and disadvantages of the evaluation methods for state-of-the art KE tools.

6.6.1 Observation

Three KE methods used in this were observed by developing a new real-world domain model called Road Traffic Accidents (RTA) and testing the domain model using state-of-the-art planners. In the core of the observation we use two broad categories of software engineering metrics called *process* and *product*.

- process - The design process has been observed to understand the operationality and efficiency of the tool or the technique. It has been noted that the design processes are completely different from each other. On the other hand, the ICKEPS did not take into account such criteria to evaluate hands on experience of the user as it is not an important criterion for competition environment. However, the process of developing domain models using different methods depends on the user's expertise, efficiency of the tool and knowledge on the modelling language. The process of developing a domain model using any tool gives more understanding as it reflects the user's experience. The user experience is more important for developing real-world domain model.

- product - The main goal of the domain modelling tools and techniques is to produce error free domain models for automated planning application. As we experienced, the method A requires more iteration to remove the entire bug from the domain model. Moreover, an error free domain model does not guarantee any valid plan which requires checking the semantic error. There are only a few tools that provide such facility to identify or reduce semantic errors. In method B, itSIMPLE UML diagrams help to keep the error level minimum. On the other hand, method B checks for error in every step of the development and does the consistency check.

  There were five RTA domain models developed by using three methods. Products developed by method A & B could be compared as they share same representation language called PDDL. The result of this evaluation is very interesting as we use same planner for the same domain model to solve same planning problem which gives different results. On the other hand, in the competition the

product (domain model) was not considered to evaluate KE tools.

6.6.2 Set Criteria

The criteria were distilled from the features extracted in Chapter 3 and from the experience of designing and developing the RTA domain model not influenced by ICKEPS competitions. The goal of a competition is to find the best tool in the selected track where our approach analyse deeper criteria. For example, debugging and maintaining two important criteria for any software development tool. Debugging is important to get error free domain model. The produced domain models may require improvement within or outside tool environment. There should be a proper guideline for maintaining the domain model by providing some kind of documentation.

6.7 Requirements for Designing Future KE Tools

The comparison of the three methods for encoding RTA domain model was fruitful. It gave us the opportunity for understanding the strengths and weakness of current KE tools and techniques. Moreover, evaluating KE tools with the set criteria and with the ICKEPS requirements give the requirements for building future KE tools.

6.7.1 Expertise

A main issue of current KE approaches for encoding domain models is that they require a specific expertise. Method A (and some approaches based on existing KE tools such as PDDL Studio) requires a PDDL expert; method B requires some expertise in UML language, which is common knowledge mainly in software engineering. Finally,

method C requires some expertise in the OCL_h language, which is not a widely known language in the AI Planning community. This requirement might significantly reduce the number of potential users of the KE tools. Users with different research background usually do not have required expertise, so they are not able to exploit existing approaches for encoding domain models. They require an expert that, due to his limited knowledge of the real world domain, will introduce some noise in the encoding. Moreover, given the hardness of generating domain models for planning, many users are not exploiting automated planning but use easier approaches, even if they are less efficient. It is also worth to consider that KE tools for encoding domain models are, usually, not very well known outside the planning community. This, again, reduces the number of users that could exploit them.

6.7.2 Team Work

Current KE tools are designed for a single user. This is usually fine; actually, the generated domain models are encoding easy domains or significantly simplified versions of complex domains. On the other hand, the number of efficient KE tools is growing, especially in the last few years. Hopefully, in coming years, we will be able to encode very accurate models of complex real world domains. In this scenario, it seems reasonable that many experts will have to cooperate for generating a domain model. From this perspective it is straightforward to consider the need of tools explicitly designed for team work as a critical requirement for future KE tools.

6.7.3 Maintenance

Users are not supported by existing KE tools for writing documentation related to the generated model. Users are usually not writing any sort of documentation. The result is that it is often quite hard to change an existing domain model after a few months. Providing a support for writing documentation, would make changes easier and would also help the users while encoding the model. The process of describing what has been done is a first test for the model. Furthermore, some tools are not able to handle domain models that have been changed manually, or by using a different tool. This limits the support that such tools could give to the life cycle of domain models.

6.7.4 Debug

We noticed that the checking tools provided by GIPO are very helpful for minimising the time spent on dynamic debug. Moreover, exploiting the automatic debug is a strategy for reducing the number of bugs that remain in the domain model, since many problems are usually not easy to find by dynamic debug. A significant improvement in the techniques for automatic debug of static/dynamic constraints will lead to significantly better encoded domain models.

6.7.5 Language support

Existing KE tools for generating domain models for planning have a very limited support of the PDDL language. Most of them are supporting only PDDL, while a few of them are also able to handle some structures of PDDL2.1 (Fox and Long, 2001). It is noticeable that the latest versions of PDDL have some features (e.g. durative actions, actions costs etc.) that are fundamental for a correct encoding of real world domains.

Furthermore, none of the existing tools support PDDL+ (Howey et al., 2004). PDDL+ provides features for capturing features of continuous planning, which are needed in systems working in real-time and that must be able to react on unexpected events.

6.8 Summary

This chapter evaluates three knowledge engineering tools and techniques by using an experimental setup on Road Traffic Accident (RTA) domain model. We also evaluate those methods by using some set criteria to understand the future needs for knowledge engineering tools development. This evaluation gives many insights on domain modelling for automated planning. It has been noted that the good quality domain model leads to the better plan generation. The experiment shows that the RTA domain encoded in PDDL exploiting method B requires less CPU time for LPG to generate plans. Moreover, the plans have a significantly shorter duration since LPG is able to effectively employ all the available emergency vehicles. This is due to a less constrained domain description that still allows the generation of legal and sound plans. These results also give some insights of the evaluation process that can play an important role to find better tools or methods for modelling real-world planning applications as well as will be helpful for developing future KE tools.

Chapter 7

Conclusion and Future Work

7.1 Summary

In this thesis 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 three methods: (i) Hand-Coding: the traditional method of a PDDL expert and text editor, (ii) itSIMPLE: a leading planning GUI with built in UML modelling tools and (iii) GIPO: an object-based notation inspired by formal 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 (KE) aspects such as how long it takes to create a model, which tools can be used to verify the model, which tool or technique uses better techniques for finding errors, what kind of error can a tool identify etc. Evaluating these three 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 engi-

neering literature - *product* (the domain model, and its use with a planner to produce plans) and *process* (the method of encoding and debugging the domain model). An important result of these evaluations was that itSIMPLE led to models developed that facilitated state-of-the-art domain-independent planners in producing good quality solutions, even though the developers were not experts in using the tool. This evaluation shows that an important line of research is to investigate coupling knowledge engineering tools and planners more tightly, so that the effects of particular design decisions influenced by the tools can be seen directly in the quality of planning solutions.

It has also been 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. Given such limitations we selected, for comparing the generated domains, only two well-known planners that can handle required features. Comparing the hand coded method (method A) and the structured method (method B) was most fruitful, as they both produced PDDL models.

The results of the comparison were illuminating. The KE based method achieved great results in both process and product metrics. From the process point of view, the method B is easy to replicate and does not require a high expertise in planning languages. From the product point of view, method B domains are easy to read and understand, while method A ones are complex to read and to maintain. Moreover, method B domains have shown to improve the performance of both the selected planners, even if on different metrics; LPG is significantly faster and SGPlan generates better quality plans.

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 exper-

imental 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 with respect to the predefined metric.

Regarding the construction of the RTA domain model in the HTN language (OCL_h) utilising GIPO, we observed that the extensive use of static tools resulted in a model that required less dynamic testing during the debugging phase, though resulting in a model of significantly larger size. An advantage of using an HTN encoding, however, is that methods compound building blocks of plans and therefore allow developers to encode intuitive plan traces.

These knowledge engineering methods have also been evaluated using a set of criteria, i.e., operationality, collaboration, maintenance, experience, efficiency, debugging and support. We highlighted the strengths and weakness of existing methods and tools and we discussed the needed in the design of future tools support for PDDL-inspired development. We observed that creating different models does not take very different amounts of time while developing the methods A and B. Method C requires significantly more time, since this method checks the consistency of the hierarchy and invariants as well as the HTN methods and tasks. Compare to method A and C, B provides better facilities to maintain the constructed domain model. The UML diagrams act as documentation that helps the user to understand and amend the domain

model if required. The hand-coded method A relies on the dynamic testing for debugging and removes bugs by a number of iterations. Method B, itSIMPLE does not provide any static testing as method A but uses a structured process which keeps the error level lower than the A. However, method C generated domain model is harder to understand as it is a different language and produces larger size domain model than the other methods, but reduces bugs in the early stage of domain development by providing static checks.

7.2 Limitations of this Research

This thesis contains a number of assumptions and limitations to restrict the scope and scale of the research which have been necessary to identify the specific project. The limitations that have been identified are -

1. The experiment has been conducted by constructing planning domain using PDDL, PDDL2.1 and OCL_h languages to enforce two leading (STRIPS and HTN) planning systems.

2. The experiments used three planners LPG, SGPlan and $HyHTN$. $HyHTN$ is currently the only available planner for object-centred HTN planner. LPG and SGPlan both can use negative pre-conditions which is necessary for real-world domain.

3. Evaluation metrics, process and project, have been used from software engineering discipline since there is no such set metrics for evaluating domain modelling tools, techniques and the model itself.

4. This thesis uses only one real-world domain, Road Traffic Accident (RTA), for constructing and evaluating knowledge engineering tools and techniques.

7.3 Future Work

This research can be extended in many different ways in the future. The future directions of the research can be summarised as below:

- Future work will involve a simulation framework for evaluating 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). Such complex domain model can be tested by using a wide range of planners.
- It would be more appropriate to 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 on unexpected events.
- We are also interested in improving the KE tools comparison by considering also other existing tools and a larger set of features to compare such as the quality of the solutions found, and run-times of different planners on generated domain models.

- Another interesting area might be to compare our centralised approach to using a multi-agent approach which moves the problem from a centralised to a distributed point of view.

- In the future work it would be worth to consider evaluating the automated tools like ARMS (Wu et al., 2005), LOCM (Cresswell et al., 2013) for knowledge engineering. The evaluation can be assisted by a number of domain features, such as, domain types, relation fluency, inconsistent effects and reversible actions (Wickler, 2011).

Appendix A: Hand-coded PDDL Domain Model

Hand-coded PDDL file for Road Traffic Accident(RTA) Domain model.

```
(define (domain accidentrecd)
(:requirements :adl)
  (:types
     ambulance police_car tow_truck fire_brigade - vehicle
     acc_victim vehicle car - subject
     city_location city - location
     accident_location hospital police_station garage
     fire_station - city_location
     route
     accident)
  (:predicates
    (at ?physical_obj1 - subject ?location1 - location)
    (available ?vehicle1 - vehicle)
    (busy ?vehicle1 - vehicle)
    (waiting ?subject1 - subject)
    (certified ?subject1 - subject)
    (aided ?subject1 - acc_victim)
    (uncertified ?subject1 - subject)
```

```
    (delivered ?subject1 - subject)
    (loaded ?subject1 - subject ?vehicle1 - vehicle)
    (identified ?accident1 - accident)
    (vehicle_involve ?vehicle1 - vehicle)
    (connects ?route1 - route ?location1 - location
    ?location2 - location)
    (in_city ?location1 - location ?city1 - city)
    (route_available ?route1 - route)
    (trapped ?hum - acc_victim)
    (on_fire ?car_acc - car )
  )
  (:action confirm_accident
       :parameters (?V - police_car ?P - subject ?A - accident_location)
       :precondition (and
    (at ?V ?A)
            (at ?P ?A)
            (uncertified ?P)
       )
       :effect (and
            (not (uncertified ?P))
            (waiting ?P)
            (certified ?P)
        )
    )
(:action untrap
       :parameters (?V - fire_brigade ?P - acc_victim
       ?A - accident_location)
       :precondition (and
            (at ?P ?A)
            (at ?V ?A)
```

```
    (available ?V)
            (certified ?P)
            (waiting ?P)
    (trapped ?P)
       )
       :effect (and
            (not (available ?V))
            (not (trapped ?P))
    (available ?V)
        )
    )
(:action extinguish_fire
       :parameters (?V - fire_brigade ?P - car ?A - accident_location)
       :precondition (and
            (at ?P ?A)
            (at ?V ?A)
    (available ?V)
            (certified ?P)
            (waiting ?P)
    (on_fire ?P)
       )
       :effect (and
            (not (available ?V))
            (not (on_fire ?P))
    (available ?V)
        )
    )
  (:action first_aid
       :parameters (?V - ambulance ?P - acc_victim ?A - accident_location )
       :precondition (and
```

```
          (at ?P ?A)
          (at ?V ?A)
          (certified ?P)
          (waiting ?P)
  (not (trapped ?P))
     )
     :effect (and
          (aided ?P)
      )
  )
(:action load_victim
     :parameters ( ?V - ambulance ?L - accident_location ?P - acc_victim)
     :precondition (and
          (at ?V ?L)
          (at ?P ?L)
          (certified ?P)
          (waiting ?P)
          (aided ?P)
  (available ?V)
     )
     :effect (and
          (not (waiting ?P))
  (not (at ?P ?L))
  (not (available ?V))
  (busy ?V)
          (loaded ?P ?V)
      )
  )
(:action move
     :parameters ( ?V - vehicle ?O - location ?City - city
```

```
       ?L - location ?City1 - city ?R - route)
       :precondition (and
(at ?V ?O)
             (in_city ?O ?City)
             (in_city ?L ?City1)
             (connects ?R ?City ?City1)
       )
       :effect (and
  (not (at ?V ?O))
                 (at ?V ?L)
        )
    )
  (:action move_in_city
       :parameters ( ?V - vehicle ?O - location ?City - city ?L - location)
       :precondition (and
            (at ?V ?O)
            (in_city ?O ?City)
            (in_city ?L ?City)
       )
       :effect (and
            (not (at ?V ?O))
            (at ?V ?L)
        )
    )
  (:action load_car
       :parameters ( ?V - tow_truck ?L - accident_location ?P - car)
       :precondition (and
            (at ?V ?L)
            (at ?P ?L)
            (waiting ?P)
```

```
          (certified ?P)
   (available ?V)
   (not (on_fire ?P))
      )
      :effect (and
          (not (available ?V))
   (busy ?V)
          (not (waiting ?P))
   (not (at ?P ?L))
          (loaded ?P ?V)
       )
   )
 (:action unload_car
      :parameters ( ?P - car ?L - garage ?V - tow_truck )
      :precondition (and
           (at ?V ?L)
           (loaded ?P ?V)
   (busy ?V)
      )
      :effect (and
          (not (loaded ?P ?V))
   (at ?P ?L)
          (not (busy ?V))
          (waiting ?P)
          (available ?V)
       )
   )
 (:action unload_victim
      :parameters ( ?P - acc_victim ?L - hospital ?V - ambulance)
      :precondition (and
```

```
            (at ?V ?L)
    (busy ?V)
            (loaded ?P ?V)
            (certified ?P)
            (aided ?P)
       )
       :effect (and
    (available ?V)
            (not (loaded ?P ?V))
    (at ?P ?L)
            (not (busy ?V))
            (waiting ?P)
            (available ?V)
        )
    )
  (:action deliver_victim
       :parameters ( ?P - acc_victim ?L - hospital )
       :precondition (and
            (at ?P ?L)
            (waiting ?P)
            (certified ?P)
            (aided ?P)
       )
       :effect (and
            (not (waiting ?P))
            (not (certified ?P))
            (not (aided ?P))
            (delivered ?P)
        )
    )
```

```
  (:action deliver_vehicle
       :parameters ( ?P - car ?L - garage )
       :precondition (and
             (at ?P ?L)
             (waiting ?P)
             (certified ?P)
       )
       :effect (and
             (not (waiting ?P))
             (not (certified ?P))
             (delivered ?P)
        )
    )
)
```

Appendix B: Hand-coded PDDL2.1 Domain Model

Hand-coded PDDL2.1 domain model for Road Traffic Accident (RTA) domain.

```
(define (domain accidentrecd)
(:requirements :typing :durative-actions)
  (:types
     ambulance police_car tow_truck fire_brigade - vehicle
     acc_victim vehicle car - subject
     city_location city - location
     accident_location hospital police_station garage fire_station - city_location
     route
     accident)
  (:predicates
    (at ?physical_obj1 - subject ?location1 - location)
    (available ?vehicle1 - vehicle)
    (busy ?vehicle1 - vehicle)
    (waiting ?subject1 - subject)
    (certified ?subject1 - subject)
    (aided ?subject1 - acc_victim)
    (uncertified ?subject1 - subject)
    (delivered ?subject1 - subject)
```

```
    (loaded ?subject1 - subject ?vehicle1 - vehicle)
    (identified ?accident1 - accident)
    (vehicle_involve ?vehicle1 - vehicle)
    (connects ?route1 - route ?location1 - location ?location2 - location)
    (in_city ?location1 - location ?city1 - city)
    (route_available ?route1 - route)
    (trapped ?hum - acc_victim)
    (on_fire ?car_acc - car )
  )
(:functions
           (distance ?O - location ?L - location)
           (route-length ?O - route)
   (confirmation-time)
   (firstaid-time)
   (speed ?V - vehicle)
   (loading-time)
   (loading-time-car)
           (unloading-time)
           (delivery-time)
   (untrapping-time)
   (extinguishing-time)
            )


  (:durative-action confirm_accident
      :parameters (?V - police_car ?P - subject ?A - accident_location)
      :duration (= ?duration (confirmation-time))
      :condition (and
    (at start (at ?V ?A))
           (at start (at ?P ?A))
           (at start (uncertified ?P))
```

```
       )
       :effect (and
            (at start (not (uncertified ?P)))
            (at end (waiting ?P))
            (at end (certified ?P))
        )
    )
(:durative-action untrap
       :parameters (?V - fire_brigade ?P - acc_victim ?A - accident_location)
:duration (= ?duration (untrapping-time))
       :condition (and
            (at start (at ?P ?A))
            (at start (at ?V ?A))
            (at start (certified ?P))
    (at start (available ?V))
            (at start (waiting ?P))
    (at start (trapped ?P))
       )
       :effect (and
    (at start (not (available ?V)))
            (at end (not (trapped ?P)))
    (at end (available ?V))
        )
    )
(:durative-action extinguish_fire
       :parameters (?V - fire_brigade ?P - car ?A - accident_location)
:duration (= ?duration (extinguishing-time))
       :condition (and
            (at start (at ?P ?A))
            (at start (at ?V ?A))
```

```
    (at start (available ?V))
            (at start (certified ?P))
            (at start (waiting ?P))
    (at start (on_fire ?P))
       )
       :effect (and
    (at start (not (available ?V)))
            (at end (not (on_fire ?P)))
    (at end (available ?V))
        )
    )
  (:durative-action first_aid
       :parameters (?V - ambulance ?P - acc_victim ?A - accident_location )
:duration (= ?duration (firstaid-time))
       :condition (and
            (at start (at ?P ?A))
            (at start (at ?V ?A))
            (at start (certified ?P))
            (at start (waiting ?P))
    (at start (not (trapped ?P)))
       )
       :effect (and
    (at start (not (waiting ?P)))
            (at end (waiting ?P))
            (at end (aided ?P))
        )
    )
  (:durative-action load_victim
       :parameters ( ?V - ambulance ?L - accident_location ?P - acc_victim)
:duration (= ?duration (loading-time))
```

```
       :condition (and
            (at start (at ?V ?L))
            (at start (at ?P ?L))
            (at start (certified ?P))
            (at start (waiting ?P))
            (at start (aided ?P))
    (at start (available ?V))
       )
       :effect (and
            (at start (not (available ?V)))
            (at start (busy ?V))
            (at start (not (waiting ?P)))
    (at start (not (at ?P ?L)))
            (at end (loaded ?P ?V))
        )
    )
  (:durative-action move
       :parameters ( ?V - vehicle ?O - location ?City - city
       ?L - location ?City1 - city ?R - route)
       :duration (= ?duration (/ (route-length ?R) (speed ?V)))
       :condition (and
(at start (at ?V ?O))
             (at start (in_city ?O ?City))
             (at start (in_city ?L ?City1))
             (at start (connects ?R ?City ?City1))
       )
       :effect (and
  (at start (not (at ?V ?O)))
                 (at end (at ?V ?L))
        )
```

```
  )
(:durative-action move_in_city
     :parameters ( ?V - vehicle ?O - location ?City - city ?L - location)
     :duration(=?duration(/(distance ?O ?L) (speed ?V)))
     :condition (and
          (at start (at ?V ?O))
          (at start (in_city ?O ?City))
          (at start (in_city ?L ?City))
     )
     :effect (and
          (at start (not (at ?V ?O)))
          (at end (at ?V ?L))
      )
  )
(:durative-action load_car
     :parameters ( ?V - tow_truck ?L - accident_location ?P - car)
     :duration (= ?duration (loading-time-car))
     :condition (and
          (at start (at ?V ?L))
          (at start (at ?P ?L))
          (at start (waiting ?P))
          (at start (certified ?P))
  (at start (available ?V))
  (at start (not (on_fire ?P)))
     )
     :effect (and
          (at start (not (available ?V)))
          (at start (busy ?V))
          (at start (not (waiting ?P)))
  (at start (not (at ?P ?L)))
```

```
          (at end (loaded ?P ?V))
       )
  )
(:durative-action unload_car
     :parameters ( ?P - car ?L - garage ?V - tow_truck )
     :duration (= ?duration (unloading-time))
     :condition (and
          (at start (at ?V ?L))
          (at start (loaded ?P ?V))
  (at start (busy ?V))
     )
     :effect (and
          (at start (not (loaded ?P ?V)))
  (at end (at ?P ?L))
          (at start (not (busy ?V)))
          (at end (waiting ?P))
          (at end (available ?V))
       )
  )
(:durative-action unload_victim
     :parameters ( ?P - acc_victim ?L - hospital ?V - ambulance)
     :duration (= ?duration (unloading-time))
     :condition (and
          (at start (at ?V ?L))
          (at start (loaded ?P ?V))
          (at start (certified ?P))
          (at start (aided ?P))
  (at start (busy ?V))
     )
     :effect (and
```

```
          (at start (not (loaded ?P ?V)))
   (at end (at ?P ?L))
          (at end (not (busy ?V)))
          (at end (waiting ?P))
          (at end (available ?V))
       )
   )
 (:durative-action deliver_victim
      :parameters ( ?P - acc_victim ?L - hospital )
      :duration (= ?duration (delivery-time))
      :condition (and
          (at start (at ?P ?L))
          (at start (waiting ?P))
          (at start (certified ?P))
          (at start (aided ?P))
      )
      :effect (and
          (at start (not (waiting ?P)))
          (at start (not (certified ?P)))
          (at start (not (aided ?P)))
          (at end (delivered ?P))
       )
   )
 (:durative-action deliver_vehicle
      :parameters ( ?P - car ?L - garage )
      :duration (= ?duration (delivery-time))
      :condition (and
          (at start (at ?P ?L))
          (at start (waiting ?P))
          (at start (certified ?P))
```

```
        )
        :effect (and
             (at start (not (waiting ?P)))
             (at start (not (certified ?P)))
             (at end (delivered ?P))
         )
    )
)
```

Appendix C: PDDL Domain Model using itSIMPLE4.0

PDDL domain model generated by itSIMPLE4.0

```
(define (domain TrafficIncident)
    (:requirements :typing :negative-preconditions :equality)
    (:types
      Physical_obj - object
      Vehicle - Physical_obj
      Rescue_v - Vehicle
      Ambulance - Rescue_v
      Tow_truck - Rescue_v
      Police_car - Vehicle
      FireService - Vehicle
      Subject - Physical_obj
      Acc_victim - Subject
      Broken_vehicle - Subject
      Human - object
      City - object
      Location - City
      City_location - Location
      Hospital - Location
```

```
  Accident_location - Location
  Police_station - Location
  Garage - Location
  Fire_Service_Station - Location
  Route - object
)
(:predicates
  (in_city ?loc - Location ?cit - City)
  (at ?phy - Physical_obj ?loc - Location)
  (moveable ?veh - Vehicle)
  (available ?veh - Vehicle)
  (busy ?veh - Vehicle)
  (vehicle_involved ?veh - Vehicle)
  (certified ?sub - Subject)
  (loaded ?sub - Subject ?vehicle1 - Vehicle)
  (unloaded ?sub - Subject)
  (delivered ?sub - Subject)
  (waiting ?sub - Subject)
  (on_fire ?sub - Subject)
  (aided ?acc - Acc_victim)
  (working ?V - FireService ?P - Broken_vehicle)
  (working_h ?V - FireService ?P - Acc_victim)
  (trapped ?acc - Acc_victim)
  (route_available ?rou - Route)
  (connects ?rou - Route ?city1 - City ?city2 - City)
)
(:action move
 :parameters (?V - Vehicle ?O - Location ?City - City
 ?L - Location ?City1 - City ?R - Route)
 :precondition
```

```
    (and
      (at ?V ?O)
      (moveable ?V)
      (in_city ?O ?City)
      (in_city ?L ?City1)
      (not (= ?City ?City1))
      (connects ?R ?City ?City1)
    )
  :effect
    (and
      (at ?V ?L)
      (not (at ?V ?O))
    )
)
(:action move_in_city
 :parameters (?V - Vehicle ?O - Location
 ?City - City ?L - Location)
 :precondition
    (and
      (at ?V ?O)
      (moveable ?V)
      (in_city ?O ?City)
      (in_city ?L ?City)
      (= ?City ?City)
    )
 :effect
    (and
      (at ?V ?L)
      (not (at ?V ?O))
    )
```

```
)
(:action first_aid
 :parameters (?V - Ambulance ?L - Accident_location
 ?P - Acc_victim)
 :precondition
   (and
     (at ?V ?L)
     (at ?P ?L)
     (certified ?P)
     (not (on_fire ?P))
     (not (trapped ?P))
   )
 :effect
   (and
     (aided ?P)
     (waiting ?P)
   )
)
(:action load_victim
 :parameters (?V - Ambulance ?L - Accident_location
 ?P - Acc_victim)
 :precondition
   (and
     (at ?V ?L)
     (at ?P ?L)
     (certified ?P)
     (waiting ?P)
     (aided ?P)
     (available ?V)
   )
```

```
 :effect
   (and
     (not (waiting ?P))
     (not (available ?V))
     (loaded ?P ?V)
     (busy ?V)
   )
)
(:action unload_victim
 :parameters (?P - Acc_victim ?L - Hospital ?V - Ambulance)
 :precondition
   (and
     (at ?V ?L)
     (loaded ?P ?V)
     (busy ?V)
   )
 :effect
   (and
     (not (loaded ?P ?V))
     (at ?P ?L)
     (not (busy ?V))
     (available ?V)
   )
)
(:action deliver_victim
 :parameters (?P - Acc_victim ?L - Hospital)
 :precondition
   (at ?P ?L)
 :effect
   (delivered ?P)
```

```
)
(:action confirm_accident
 :parameters (?V - Police_car ?P - Subject
 ?A - Accident_location)
 :precondition
   (and
     (at ?V ?A)
     (at ?P ?A)
     (not (certified ?P))
   )
 :effect
   (and
     (at ?V ?A)
     (at ?P ?A)
     (certified ?P)
   )
)
(:action load_broken_vehicle
 :parameters (?V - Tow_truck ?L - Accident_location
 ?P - Broken_vehicle)
 :precondition
   (and
     (at ?V ?L)
     (at ?P ?L)
     (certified ?P)
     (not (on_fire ?P))
     (available ?V)
   )
 :effect
   (and
```

```
        (not (waiting ?P))
        (not (at ?P ?L))
        (loaded ?P ?V)
        (not (available ?V))
      )
    )
    (:action unload_broken_vehicle
     :parameters (?P - Broken_vehicle ?L - Garage ?V - Tow_truck)
     :precondition
       (and
        (at ?V ?L)
        (loaded ?P ?V)
       )
     :effect
       (and
        (not (loaded ?P ?V))
        (at ?P ?L)
        (not (busy ?V))
        (available ?V)
       )
    )
    (:action deliver_broken_vehicle
     :parameters (?P - Broken_vehicle ?L - Garage)
     :precondition
       (at ?P ?L)
     :effect
       (delivered ?P)
    )
(:action start_rescue_victim
       :parameters (?V - FireService ?P - Acc_victim
```

```
       ?A - Accident_Location)
       :precondition (and
            (at ?P ?A)
            (at ?V ?A)
    (available ?V)
            (certified ?P)
    (trapped ?P)
       )
       :effect (and
            (not (available ?V))
    (working_h ?V ?P)
        )
    )
(:action finish_rescue_victim
       :parameters (?V - FireService ?P - Acc_victim
       ?A - Accident_Location)
       :precondition (and
            (at ?P ?A)
            (at ?V ?A)
            (certified ?P)
    (trapped ?P)
    (working_h ?V ?P)
       )
       :effect (and
            (not (trapped ?P))
    (available ?V)
    (not (working_h ?V ?P))
        )
    )
(:action start_extinguish_fire
```

```
       :parameters (?V - FireService ?P - Broken_vehicle
       ?A - Accident_location)
       :precondition (and
            (at ?P ?A)
            (at ?V ?A)
    (available ?V)
            (certified ?P)
    (on_fire ?P)
       )
       :effect (and
            (not (available ?V))
    (working ?V ?P)
        )
    )
(:action finish_extinguish_fire
       :parameters (?V - FireService ?P - Broken_vehicle
       ?A - Accident_location)
       :precondition (and
            (at ?P ?A)
            (at ?V ?A)
            (certified ?P)
    (on_fire ?P)
    (working ?V ?P)
       )
       :effect (and
    (not (working ?V ?P))
            (not (on_fire ?P))
    (available ?V)
        )
    )
```

)

Appendix D: PDDL2.1 Domain Model using itSIMPLE4.0

Road Traffic Accident (RTA) domain model developed by itSIMPLE in PDDL2.1

```
(define (domain TrafficIncident)
    (:requirements :typing :negative-preconditions :equality :durative-actions)
    (:types
      Physical_obj - object
      Vehicle - Physical_obj
      Rescue_v - Vehicle
      Ambulance - Rescue_v
      Tow_truck - Rescue_v
      Police_car - Vehicle
      FireService - Vehicle
      Subject - Physical_obj
      Acc_victim - Subject
      Broken_vehicle - Subject
      Human - object
      City - object
      Location - City
      City_location - Location
      Hospital - Location
```

```
      Accident_location - Location
      Police_station - Location
      Garage - Location
      Fire_Service_Station - Location
      Route - object
    )
    (:predicates
      (in_city ?loc - Location ?cit - City)
      (at ?phy - Physical_obj ?loc - Location)
      (moveable ?veh - Vehicle)
      (available ?veh - Vehicle)
      (busy ?veh - Vehicle)
      (vehicle_involved ?veh - Vehicle)
      (certified ?sub - Subject)
      (loaded ?sub - Subject ?vehicle1 - Vehicle)
      (unloaded ?sub - Subject)
      (delivered ?sub - Subject)
      (waiting ?sub - Subject)
      (on_fire ?sub - Subject)
      (aided ?acc - Acc_victim)
      (trapped ?acc - Acc_victim)
      (route_available ?rou - Route)
      (connects ?rou - Route ?city1 - City ?city2 - City)
    )
(:functions
           (distance ?O - location ?L - location)
           (route-length ?O - route)
   (confirmation-time)
   (firstaid-time)
   (speed ?V - vehicle)
```

```
(loading-time)
(loading-time-car)
        (unloading-time)
        (delivery-time)
(untrapping-time)
(extinguishing-time)
         )

 (:durative-action move
  :parameters (?V - Vehicle ?O - Location ?City - City
  ?L - Location ?City1 - City ?R - Route)
    :duration (= ?duration (/ (route-length ?R) (speed ?V)))
  :condition
    (and
      (at start (at ?V ?O))
      (at start (moveable ?V))
      (at start (in_city ?O ?City))
      (at start (in_city ?L ?City1))
      (at start (not (= ?City ?City1)))
      (at start (connects ?R ?City ?City1))
    )
  :effect
    (and
      (at end (at ?V ?L))
      (at start (not (at ?V ?O)))
    )
 )
 (:durative-action move_in_city
  :parameters (?V - Vehicle ?O - Location ?City - City
  ?L - Location)
```

```
      :duration(=?duration(/(distance ?O ?L) (speed ?V)))
    :condition
      (and
        (at start (at ?V ?O))
        (at start (moveable ?V))
        (at start (in_city ?O ?City))
        (at start (in_city ?L ?City))
        (at start (= ?City ?City))
      )
    :effect
      (and
        (at end (at ?V ?L))
        (at start (not (at ?V ?O)))
      )
  )
  (:durative-action first_aid
   :parameters (?V - Ambulance ?L - Accident_location
   ?P - Acc_victim)
:duration (= ?duration (firstaid-time))
    :condition
      (and
        (at start (at ?V ?L))
        (at start (at ?P ?L))
        (at start (certified ?P))
        (at start (not (on_fire ?P)))
        (at start (not (trapped ?P)))
      )
    :effect
      (and
        (at end (aided ?P))
```

```
          (at end (waiting ?P))
        )
    )
    (:durative-action load_victim
     :parameters (?V - Ambulance ?L - Accident_location
     ?P - Acc_victim)
:duration (= ?duration (loading-time))
     :condition
       (and
         (at start (at ?V ?L))
         (at start (at ?P ?L))
         (at start (certified ?P))
         (at start (waiting ?P))
         (at start (aided ?P))
         (at start (available ?V))
       )
     :effect
       (and
         (at start (not (waiting ?P)))
         (at start (not (available ?V)))
         (at end (loaded ?P ?V))
         (at start (not (at ?P ?L)))
       )
    )
    (:durative-action unload_victim
     :parameters (?P - Acc_victim ?L - Hospital ?V - Ambulance)
       :duration (= ?duration (unloading-time))
     :condition
       (and
         (at start (at ?V ?L))
```

```
      (at start (loaded ?P ?V))
      (at start (not (available ?V)))
    )
  :effect
    (and
      (at end (not (loaded ?P ?V)))
      (at end (at ?P ?L))
      (at end (available ?V))
    )
)
(:durative-action deliver_victim
 :parameters (?P - Acc_victim ?L - Hospital)
   :duration (= ?duration (delivery-time))
 :condition
   (at start (at ?P ?L))
 :effect
   (at end (delivered ?P))
)
(:durative-action confirm_accident
 :parameters (?V - Police_car ?P - Subject ?A - Accident_location)
   :duration (= ?duration (confirmation-time))
 :condition
   (and
     (at start (at ?V ?A))
     (at start (at ?P ?A))
     (at start (not (certified ?P)))
   )
 :effect
   (and
     (at start (at ?V ?A))
```

```
      (at start (at ?P ?A))
      (at end (certified ?P))
    )
)
(:durative-action load_broken_vehicle
 :parameters (?V - Tow_truck ?L - Accident_location
 ?P - Broken_vehicle)
   :duration (= ?duration (loading-time-car))
 :condition
   (and
     (at start (at ?V ?L))
     (at start (at ?P ?L))
     (at start (certified ?P))
     (at start (not (on_fire ?P)))
     (at start (available ?V))
   )
 :effect
   (and
     (at start (not (waiting ?P)))
     (at start (not (at ?P ?L)))
     (at end (loaded ?P ?V))
     (at start (not (available ?V)))
   )
)
(:durative-action unload_broken_vehicle
 :parameters (?P - Broken_vehicle ?L - Garage ?V - Tow_truck)
   :duration (= ?duration (unloading-time))
 :condition
   (and
     (at start (at ?V ?L))
```

```
          (at start (loaded ?P ?V))
        )
      :effect
        (and
          (at end (not (loaded ?P ?V)))
          (at end (at ?P ?L))
          (at end (not (busy ?V)))
          (at end (available ?V))
        )
    )
    (:durative-action deliver_broken_vehicle
      :parameters (?P - Broken_vehicle ?L - Garage)
        :duration (= ?duration (delivery-time))
      :condition
        (at start (at ?P ?L))
      :effect
        (at end (delivered ?P))
    )
(:durative-action rescue_victim
        :parameters (?V - FireService ?P - Acc_victim
        ?A - Accident_Location)
:duration (= ?duration (untrapping-time))
        :condition (and
    (at start (at ?P ?A))
            (at start (at ?V ?A))
    (at start (available ?V))
            (at start (certified ?P))
    (at start (trapped ?P))
        )
        :effect (and
```

```
            (at start (not (available ?V)))
            (at end (not (trapped ?P)))
    (at end (available ?V))
        )
    )
(:durative-action extinguish_fire
       :parameters (?V - FireService ?P - Broken_vehicle
       ?A - Accident_location)
:duration (= ?duration (extinguishing-time))
       :condition (and
          (at start  (at ?P ?A))
          (at start  (at ?V ?A))
  (at start  (available ?V))
          (at start  (certified ?P))
  (at start  (on_fire ?P))
       )
       :effect (and
            (at start (not (available ?V)))
    (at end (not (on_fire ?P)))
    (at end (available ?V))
        )
    )
)
```

Appendix E: OCL_h Domain Model using GIPO-III

Road Traffic Domain Model in OCL_h, generated by GIPO-III.

```
/**
 *  All rights reserved. Use of this software is permitted for
 *non-commercial research purposes, and it may be copied only for that use.
 * All copies must include this copyright message.  This software
 is made available AS IS, and
 *  neither the GIPO team nor the University of Huddersfield make any warranty
 *  about the software or its performance.
 *
 *  Automatically generated OCL Domain from  GIPO Version 3.0
 *
 *  Author: Shahin Shah
 *  Institution: University of Huddersfield
 *  Date created: August 2011
 *  Date last modified: 2013/01/10 at 05:26:33 PM GMT
 *  Description:
 *    The main goal of this domain is to model road traffic domain
 in Ainley top roundabout
 *    There is an area  West Yorkshire with 8 different
```

```
 zones(Ainley top, Huddersfield, Queensbury, bradley,
  Brighouse, Greetland, Bailiff Bridge and Halifax) where
 *    each zone has number of location where accident
 can happen.
 *    There are a number of service vehicle available to
 take accident victims to hospital,
 *    broken vehicle to recovery center and police station
  to take the suspect.
 */

domain_name(rta).

option(hierarchical).

% Sorts
sorts(primitive_sorts,[ambulance,tow_truck,police_car,acc_victim,
broken_vehicle,accident_location,police_station,hospital,
garage,aclocation,route]).
sorts(physical_obj,[vehicle,subject]).
sorts(vehicle,[rescue_v,police_car]).
sorts(rescue_v,[ambulance,tow_truck]).
sorts(subject,[acc_victim,broken_vehicle]).
sorts(city,[location]).
sorts(location,[accident_location,police_station,hospital,garage]).

% Objects
objects(ambulance,[ambu_hud,ambu_hal,ambu_brig]).
objects(tow_truck,[tow_truck_halifax,tow_truck_huddersfield,
tow_truck_brighouse,tow_truck_queensbury]).
objects(police_car,[police_car_queen,police_car_bradley,
```

```
police_car_halifax,police_car_huddersfield]).
objects(accident_location,[accident_location_1,accident_location_2,
accident_location_3,accident_location_4]).
objects(police_station,[police_halifax,police_huddersfield,
police_queen,police_bradley]).
objects(hospital,[huddersfield_hospital,
halifax_hospital,brighouse_hospital]).
objects(garage,[garage_brighouse,
garage_huddersfield,garage_halifax,garage_queensbury]).
objects(route,[ainley_greet,bradley_ainley,
ainley_brigh,greet_halifax,
brigh_baliff,a629,brigh_queen,baliff_halifax,
hud_brigh,queen_halifax,
hud_bradley,ainley_halifax]).

% Predicates
predicates([
    at(physical_obj,location),
    moveable(vehicle),
    available(vehicle),
    busy(vehicle),
    waiting(subject),
    certified(subject),
    uncertified(subject),
    delivered(subject),
    loaded(subject,vehicle),
    identify(accident),
    type_of_accident(accident,accident_type),
    vehicle_involve(vehicle),
    connects(route,city,city),
```

```
    in_city(location,city),
    in_area(location,city_location),
    route_available(route)]).

% Object Class Definitions
substate_classes(physical_obj,P,[
    [at(P,L)]]).
substate_classes(vehicle,T,[
    [moveable(T),available(T)],
    [moveable(T),busy(T)]]).
substate_classes(subject,P,[
    [uncertified(P)],
    [waiting(P),certified(P)],
    [loaded(P,V),certified(P)],
    [delivered(P)]]).

% Atomic Invariants
atomic_invariants([
    in_city(accident_location_1,ainley_top),
    in_city(accident_location_3,ainley_top),
    in_city(accident_location_4,ainley_top),
    in_city(police_huddersfield,huddersfield),
    in_city(huddersfield_hospital,huddersfield),
    in_city(garage_huddersfield,huddersfield),
    in_city(huds_fire_station,huddersfield),
    in_city(halifax_hospital,halifax),
    in_city(garage_halifax,halifax),
    in_city(police_halifax,halifax),
    in_city(police_queen,queensbury),
    in_city(garage_queensbury,queensbury),
```

```
in_city(accident_location_2,ainley_top),
in_city(accident_location_3,greetland),
in_city(police_bradley,bradley),
in_city(brighouse_hospital,brighouse),
route_available(ainley_greet),
connects(ainley_greet,ainley_top,greetland),
connects(ainley_greet,greetland,ainley_top),
route_available(bradley_ainley),
connects(bradley_ainley,bradley,ainley_top),
connects(bradley_ainley,ainley_top,bradley),
route_available(ainley_brigh),
connects(ainley_brigh,ainley_top,brighouse),
connects(ainley_brigh,brighouse,ainley_top),
route_available(greet_halifax),
connects(greet_halifax,greetland,halifax),
connects(greet_halifax,halifax,greetland),
route_available(brigh_baliff),
connects(brigh_baliff,brighouse,bailiff_bridge),
connects(brigh_baliff,bailiff_bridge,brighouse),
route_available(a629),
connects(a629,huddersfield,ainley_top),
connects(a629,ainley_top,huddersfield),
route_available(brigh_queen),
connects(brigh_queen,brighouse,queensbury),
connects(brigh_queen,queensbury,brighouse),
route_available(baliff_halifax),
connects(baliff_halifax,halifax,bailiff_bridge),
connects(baliff_halifax,bailiff_bridge,halifax),
route_available(hud_brigh),
connects(hud_brigh,huddersfield,brighouse),
```

```
    connects(hud_brigh,brighouse,huddersfield),
    route_available(queen_halifax),
    connects(queen_halifax,halifax,queensbury),
    connects(queen_halifax,queensbury,halifax),
    route_available(hud_bradley),
    connects(hud_bradley,huddersfield,bradley),
    connects(hud_bradley,bradley,huddersfield),
    route_available(ainley_halifax),
    connects(ainley_halifax,ainley_top,halifax),
    connects(ainley_halifax,halifax,ainley_top)]).

% Implied Invariants
implied_invariant([loaded(P,V)],[at(V,L),at(P,L)]).

% Inconsistent Constraints

% Operators
operator(confirm_accident(P,A),
    % prevail
    [],
    % necessary
    [     sc(subject,P,[at(P,A),uncertified(P)]=>
    [at(P,A),waiting(P),certified(P)])],
    % conditional
    []).
operator(first_aid(P,A),
    % prevail
    [],
    % necessary
    [     sc(subject,P,[at(P,A),uncertified(P)]=>[at(P,A),
```

```
    waiting(P),certified(P)])],
    % conditional
    []).
operator(move(V,O,City,L,City1,R),
    % prevail
    [],
    % necessary
    [     sc(vehicle,V,[at(V,O),is_of_sort(R,route),moveable(V),
    in_city(O,City),in_city(L,City1),ne(City,City1),
    connects(R,City,City1)]=>[at(V,L)])],
    % conditional
    [     sc(subject,X,[loaded(X,V),certified(X),
    at(X,O)]=>[loaded(X,V),
    certified(X),at(X,L)])]).
operator(move_in_city(V,O,City,L),
    % prevail
    [],
    % necessary
    [     sc(rescue_v,V,[at(V,O),moveable(V),in_city(O,City),
    in_city(L,City)]=>[at(V,L)])],
    % conditional
    [     sc(subject,X,[loaded(X,V),certified(X),at(X,O)]=>
    [loaded(X,V),certified(X),at(X,L)])]).
operator(commission(V),
    % prevail
    [],
    % necessary
    [     sc(vehicle,V,[moveable(V),available(V)]=>
    [moveable(V),busy(V)])],
    % conditional
```

```
    []).
operator(load_subject(V,L,P),
    % prevail
    [     se(vehicle,V,[at(V,L)])],
    % necessary
    [     sc(subject,P,[at(P,L),waiting(P),certified(P)]=>
    [at(P,L),loaded(P,V),certified(P)])],
    % conditional
    []).
operator(unload_subject(P,L,V),
    % prevail
    [],
    % necessary
    [     sc(subject,P,[at(P,L),loaded(P,V),certified(P)]=>
    [at(P,L),waiting(P),certified(P)]),
     sc(vehicle,V,[at(V,L),moveable(V),busy(V)]=>
     [at(V,L),moveable(V),available(V)])],
    % conditional
    []).
operator(deliver(P,L),
    % prevail
    [],
    % necessary
    [     sc(subject,P,[at(P,L),waiting(P),certified(P)]=>
    [at(P,L),delivered(P)])],
    % conditional
    []).

% Methods
/****
```

```
 * Carry within same city
 */
method(carry_direct(P,O,D),
    % pre-condition
    [
],
    % Index Transitions
    [
     sc(subject,P,[at(P,O),waiting(P),certified(P)]=>
     [at(P,D),waiting(P),certified(P)])],
    % Static
    [
     is_of_sort(P,subject),
     is_of_sort(V,rescue_v),
     in_city(O,City),
     in_city(D,City)],
    % Temporal Constraints
    [
     before(1,2),
     before(2,3),
     before(3,4),
     before(4,5)],
    % Decomposition
    [
     commission(V),
     achieve(ss(rescue_v,V,[at(V,O)])),
     load_subject(V,O,P),
     unload_subject(P,D,V),
     move_in_city(V,O,City,D)]
).
```

```
/****
 * Carry between zones by rescue_v
 */
method(carry_direct(P,O,D),
    % pre-condition
    [
],
    % Index Transitions
    [
     sc(subject,P,[at(P,O),waiting(P),certified(P)]=>
     [at(P,D),waiting(P),certified(P)])],
    % Static
    [
     is_of_sort(P,subject),
     is_of_sort(V,rescue_v),
     in_city(O,CY),
     in_city(D,CY1),
     ne(CY,CY1),
     connects(R,CY,CY1),
     is_of_sort(R,route),
     route_available(R)],
    % Temporal Constraints
    [
     before(1,2),
     before(2,3),
     before(3,4),
     before(4,5)],
    % Decomposition
    [
     commission(V),
```

```
     achieve(ss(rescue_v,V,[at(V,O)])),
     load_subject(V,O,P),
     move(V,O,CY,D,CY1,R),
     unload_subject(P,D,V)]
).
/****
 *
 */
method(transport(Subject,Org,Dest),
    % pre-condition
    [
],
    % Index Transitions
    [
     sc(subject,Subject,[uncertified(Subject),
     at(Subject,Org)]=>[delivered(Subject),at(Subject,Dest)])],
    % Static
    [
     in_region(Org,Region),
     in_region(Dest,Region)],
    % Temporal Constraints
    [
     before(1,2),
     before(2,3)],
    % Decomposition
    [
     achieve(ss(subject,Subject,[waiting(Subject),
     certified(Subject), at(Subject,Org)])),
     carry_direct(Subject,Org,Dest),
     deliver(Subject,Dest)]
```

```
).

% Domain Tasks

% HTN Domain Tasks
htn_task(1,
    goal(
          [
            transport(acc_victim_1,accident_location_1,
            huddersfield_hospital)],
    % Temporal Constraints
          [
],
    % Static constraints
          [
]),
    % INIT States
    [
     ss(subject,acc_victim_1,[at(acc_victim_1,accident_location_1),
     uncertified(acc_victim_1)]),
     ss(ambulance,ambu_hud,[at(ambu_hud,accident_location_1),
     moveable(ambu_hud),available(ambu_hud)])]).
htn_task(2,
    goal(
          [
            transport(acc_victim_1,accident_location_1,
            huddersfield_hospital),
            transport(acc_victim_2,accident_location_1,
            halifax_hospital),
            transport(acc_victim_3,accident_location_1,
```

```
            bradford_hospital)],
    % Temporal Constraints
          [
],
    % Static constraints
          [
]),
    % INIT States
    [
     ss(subject,acc_victim_1,[at(acc_victim_1,accident_location_1),
     uncertified(acc_victim_1)]),
     ss(subject,acc_victim_2,[at(acc_victim_2,accident_location_1),
     uncertified(acc_victim_2)]),
     ss(subject,acc_victim_3,[at(acc_victim_3,accident_location_1),
     uncertified(acc_victim_3)]),
     ss(rescue_v,ambu_1,[at(ambu_1,huddersfield_hospital),
     moveable(ambu_1),available(ambu_1)]),
     ss(rescue_v,ambu_3,[at(ambu_3,halifax_hospital),moveable(ambu_3),
     available(ambu_3)]),
     ss(rescue_v,ambu_2,[at(ambu_2,bradford_hospital),moveable(ambu_2),
     available(ambu_2)])]).
htn_task(3,
    goal(
          [
            transport(bv_1,accident_location_1,garage_1),
            transport(bv_2,accident_location_1,garage_2),
            transport(bv_3,accident_location_1,garage_3)],
    % Temporal Constraints
          [
],
```

```
    % Static constraints
          [
]),
    % INIT States
    [
     ss(subject,bv_1,[at(bv_1,accident_location_1),uncertified(abv_1)]),
     ss(subject,bv_2,[at(bv_2,accident_location_1),uncertified(bv_2)]),
     ss(subject,bv_3,[at(bv_3,accident_location_1),uncertified(bv_3)]),
     ss(rescue_v,tow_truck_1,[at(tow_truck_1,accident_location_1),
     moveable(tow_truck_1),available(tow_truck_1)]),
     ss(rescue_v,tow_truck_2,[at(tow_truck_2,accident_location_1),
     moveable(tow_truck_2),available(tow_truck_2)]),
     ss(rescue_v,tow_truck_3,[at(tow_truck_3,accident_location_1),
     moveable(tow_truck_3),available(tow_truck_3)])]).
```

Bibliography

AIPS-98 Planning Competition Committee (1998). PDDL - The Planning Domain Definition Language, *Technical Report CVC TR-98-003/DCS TR-1165*, Yale Center for Computational Vision and Control. 25

Bacchus, F. (2001). AIPS-00 Planning Competition, *AI Magazine* **22**(3): 47–56. 2, 13

Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T. et al. (2012). Europa: A platform for ai planning, scheduling, constraint programming, and optimization, *Proceedings of the 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)* . 4, 24, 30, 51

Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T. et al. (n.d.). Europa: A platform for ai planning, scheduling, constraint programming, and optimization. 29

Bartak, R., Fratini, S. and McCluskey, T. (2010). The third competition on knowledge engineering for planning and scheduling, *AI Magazine* **31**(1): 95–98.
URL: *http://eprints.hud.ac.uk/7789/* 31, 34, 140

Bartak, R. and McCluskey, T. (2006). The first competition on knowledge engineering

for planning and scheduling, *AI Magazine* **Spring**: 97–98.

URL: *http://eprints.hud.ac.uk/579/* 140

Benesch (2011). *Traffic Incident Management Operations Guidelines*, Iowa Department of Transportation, Faderal Highway Administration, 1200 New jersey Avenue, SE, Washington, DC 20590. 68

Bernardi, G., Cesta, A., Orlandini, A. and Finzi, A. (2013). A knowledge engineering environment for p&s with timelines, *In Proceedings of Twenty-third International Conference on Automated Planning and Scheduling (ICAPS-13), Workshop on Knowledge Engineering for Planning and Scheduling (KEPS-13)*. 54

Berners-Lee, T., Hendler, J. and Lassila, O. (2001). The Semantic Web, *Scientific American* . 45

Bienkowski, M. (1995). Demonstrating the operational feasibility of new technologies: The ARPI IFDs, *IEEE Expert: Intelligent Systems and Their Applications* **10**: 27–33. 67

Biundo, S., Aylett, R., Beetz, M., Borrajo, D., Cesta, A., Grant, T., McCluskey, T., Milani, A. and Verfaillie, G. (2003). PLANET technological roadmap on AI planning and scheduling, Electronically avaliable at http://www.planet-noe.org/service/Resources/Roadmap/Roadmap2.pdf. xii, 30, 32

Biundo, S., Barrajo, D. and McCluskey, T. L. (2002). Planning and Scheduling: A Technology for Improving Flexibility in e-Commerce and Electronic Work, *Proceedings of e2002, The eBusiness and eWork Annual Conference, Prague The Czech Republic*. 4, 19

Biundo, S. and Schattenberg, B. (2001). From abstract crisis to concrete relief: A preliminary report on combining state abstraction and htn planning, *Proceedings of the 6th European Conference on Planning (ECP-01)*, Springer Verlag, pp. 157–168. 67

Blum, A. L. and Furst, M. L. (1997a). Fast planning through Planning Graph Analysis, *Artificial Intelligence* **90**: 281–300. 2, 13

Blum, A. L. and Furst, M. L. (1997b). Fast planning through planning graph analysis, *Artificial Intelligence* **90**: 281 – 300. 16

Bonet, B., Loerincs, G. and Geffner, H. (1997). A robust and fast action selection mechanism for planning, *In Proceedings of AAAI-97*, MIT Press, pp. 714–719. 14

Booch, G., Rumbaugh, J. and Jacobson, I. (1997). *The UML specification documents* , Rational Software Corp, Santa Clara, CA . 8, 40, 42, 94

Bouillet, E., Feblowitz, M., Feng, H., Ranganathan, A., Riabov, A., Udrea, O. and Liu, Z. (2009). Mario: middleware for assembly and deployment of multi-platform flow-based applications, *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, Springer-Verlag New York, Inc., New York, NY, USA, pp. 26:1–26:7.
URL: *http://dl.acm.org/citation.cfm?id=1656980.1657015* 53

Bresina, J. L., Jónsson, A. K., Morris, P. H. and Rajan, K. (2005). Activity planning for the Mars Exploration Rovers, *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, Monterey, California, USA, pp. 40 – 49. 1

Bresina, J., Meuleauy, N., Ramakrishnan, S., Smith, D. and Washingtonx, R. (2002).

Planning under continuous time and resource uncertainty: A challenge for AI, *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 77–84. 23

Carloni, L., Passerore, R., Pinto, A. and Sangiovanni-Vincentelli, A. (2006). Languages and tools for hybrid systems design, *Foundations and Trends in Design Automation* **1**: 1 – 204. 23

Castillo, L., Fdez-olivares, J., scar Garca-prez and Palao, F. (2006). Efficiently handling temporal knowledge in an htn planner, *In Sixteenth International Conference on Automated Planning and Scheduling, ICAPS*, AAAI, pp. 63–72. 24, 52

Chen, Y., Wah, B. W. and Hsu, C.-W. (2006). Temporal planning using subgoal partitioning and resolution in sgplan, *Journal of Artificial Intelligence Research* **26**(1): 323–369. 14, 17, 84

Chien, S., Hill, R., Wang, X., Estlin, T., Fayyad, K. and Mortenson, H. (1996). Why Real-World Planning is Difficult: A Tale of Two Applications, *in* M. Ghallab and A. Milani (eds), *New Directions in AI Planning*, IOS Press, pp. 287–298 . 3

Cresswell, S., Fox, M. and Long, D. (2002). Extending TIM domain analysis to handle ADL constructs, *in* T. L. McCluskey (ed.), *AIPS '02 Workshop on Knowledge Engineering Tools and Techniques for A.I. Planning*. 20

Cresswell, S., McCluskey, T. and West, M. M. (2013). Acquiring planning domain models using locm, *Knowledge Engineering Review* .
URL: *http://eprints.hud.ac.uk/9052/* 154

Currie, K., Tate, A. and Bridge, S. (1991). O-plan: the open planning architecture, *Artificial Intelligence* **52**: 49–86. 67

de la Asunción, M., Castillo, L. A., Fernández-Olivares, J., García-Pérez, Ó., González, A. and Palao, F. (2005). SIADEX: An interactive knowledge-based planner for decision support in forest fire fighting., *AI Communications* **18**(4): 257 – 268. 67

Drummond, M. (1994). On precondition achievement and the computational economics of automatic planning, *in* C. Bäckström and E. Sandewall (ed.), *Current Trends in AI Planning*, IOS Press. 3

Edelkamp, S. and Hoffmann, J. (2004). Pddl2.2: The language for the classical part of the 4th international planning competition, *4th International Planning Competition (IPC17), at ICAPS17* . 26

Erol, K. (1995). *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*, PhD thesis, Department of Computer Science, University of Maryland. 2, 13

Erol, K., Hendler, J. and Nau, D. S. (1994). UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning, *Proceedings of AIPS*. 3

Eyerich, P., Keller, T. and Nebel, B. (2010). Combining action and motion planning via semantic attachments, *The 20th International Conference on Automated Planning and Scheduling – Workshop on Combining Action and Motion Planning*. 26

Feblowitz, M. D., Ranganathan, A., Riabov, A. V. and Udrea, O. (2012). Planning-based composition of stream processing applications, *and Exhibits* p. 5. 53

Fernández-Olivares, J., Castillo, L. A., García-Pérez, Ó. and Palao, F. (2006). Bringing users and planning technology together: experiences in SIADEX., *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*, Cumbria, UK, pp. 11 – 20. 26, 67

Fikes, R. E. and Nilsson, N. J. (1971a). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence* **2**. 2, 12, 15, 25, 26

Fikes, R. E. and Nilsson, N. J. (1971b). Strips: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2**: 189–208. 38

Fox, M., Gerevini, A., Long, D. and Serina, I. (2006). Plan stability: Replanning versus plan repair, *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pp. 212 – 221. 16

Fox, M. and Long, D. (2001). PDDL2.1: An extension to PDDL for expressing temporal planning domains , *Technical Report, Dept of Computer Science, University of Durham*. 3, 24, 26, 44, 83, 147

Fox, M. and Long, D. (2006). Modelling mixed discrete-continuous domains for planning., *Journal of Artificial Intelligence Research* **27**: 235 – 297. 26

Fox, M., Long, D. and Magazzeni, D. (2011). Automatic construction of efficient multiple battery usage policies, *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*, pp. 74 – 81.
URL: *http://www.aaai.org/ocs/index.php/ICAPS/ICAPS11/paper/view/2683* 1, 26

Gerevini, A. and Long, D. (2005). Plan constraints and preferences in pddl3, *The Lan-*

guage of the Fifth International Planning Competition. Tech. Rep. Technical Report, Department of Electronics for Automation, University of Brescia, Italy . 26

Gerevini, A., Saetti, A. and Serina, I. (2003). Planning through stochastic local search and temporal action graphs, *Journal of Artificial Intelligence Research (JAIR)* **20**: 239 – 290. 14, 84

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D. and Wilkins, D. (1998). PDDL - the planning domain definition language, *Technical Report CVC TR-98-003/DCS TR-1165*, Yale Center for Computational Vision and Control. 3, 38, 83

Ghallab, M., Nau, D. and Traverso, P. (2004). *Automated Planning: Theory & Practice*, Morgan Kaufmann Publishers Inc. 2, 17

González-Ferrer, A., Fernández-Olivares, J. and Castillo, L. (2009). JABBAH: A java application framework for the translation between business process models and htn, *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)*, pp. 28–37. 1, 29, 52, 89

HA (2009). *Highways Agency Network Management Manual*, 5.10 edn, Highways Agency, Network Services, Network Management Policy Team, City Tower, Piccadilly Plaza, MANCHESTER M1 4BE. 66, 68

Helmert, M. (2006). The fast downward planning system, *Journal of Artificial Intelligence Research* **26**(1): 191–246. 16

Hoffman, J. and Edelkamp, S. (2005). The deterministic part of IPC-4: An overview, *Journal of Artificial Intelligence Research* **24**: 519 – 579. 16

Hoffmann, J. (2003). The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables, *Journal of Artificial Intelligence Research (JAIR)* **20**: 291–341. 84

Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search, *J. Artif. Intell. Res. (JAIR)* . 13, 16

Hoffmann, J. et al. (2003). The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables, *J. Artif. Intell. Res. (JAIR)* **20**: 291–341. 14, 16

Howey, R., Long, D. and Fox, M. (2004). Automatic plan validation, continuous effects and mixed initiative planning using PDDL., *Proceedings of the Sixteenth International Conference on Tools with Artificial Intelligence*, pp. 294 – 301. 3, 22, 26, 58, 148, 153

Hsu, C.-W. and Wah, B. W. (2008). The sgplan planning system in ipc-6, *Sixth International Planning Competition, Sydney, Australia (Sepember 2008)* . 17

J. Vodrka, L. C. (2010). Visual design of planning domains, *KEPS 2010: Workshop on Knowledge Engineering for Planning and Scheduling*. 53

Kambhampati, S. (1994). Comparing Partial Order Planning and Task Reduction Planning: A preliminary report, *Technical Report TR 94-001*, Arizona State University, Dept. of Computer Science and and Engineering. 13

Kambhampati, S. (1997). Refinement planning as a unifying framework for plan synthesis, *Artificial Intelligence* **18**. 3

Kautz, H. (2006). Deconstructing planning as satisfiability, *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, Vol. 21, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, p. 1524. 2, 14

Kautz, H. and Selman, B. (1996). Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search, *Proceedings of the Fourteenth National Conference on Artificial Intelligence*. 2, 13

Kautz, H. and Selman, B. (1999). Unifying SAT-based and Graph-based plan, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. 2

Kitchin, D. E. (2000). *Object-Centred Generative Planning*, PhD thesis, School of Computing and Mathematics, University of Huddersfield. 27

McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence, *Machine Intelligence* **4**. 12

McCluskey, T. (2000a). Knowledge engineering for planning roadmap. **URL:** *http://eprints.hud.ac.uk/8134/* 19, 20, 21, 23, 29, 30, 36

McCluskey, T. L. (2000b). Object Transition Sequences: A New Form of Abstraction for HTN Planners, *The Fifth International Conference on Artificial Intelligence Planning Systems*. 27

McCluskey, T. L. (2002). Knowledge Engineering: Issues for the AI Planning Community (Keynote Talk), *Proceedings of the AIPS-2002 Workshop on Knowledge Engineering Tools and Techniques for AI Planning, Toulouse, France*. 5

McCluskey, T. L. and Kitchin, D. E. (1998). A tool-supported approach to engineering htn planning models, *In Proceedings of 10th IEEE International Conference on Tools with Artificial Intelligence*. 27, 46, 49, 100, 106

McCluskey, T. L., Liu, D. and Simpson, R. M. (2003). GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment, *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, AAAI Press, pp. 92 – 101. 4, 18, 19, 49, 109

McCluskey, T. L. and Simpson, R. (2006). Combining constraint-based and classical formulations for planning domains: GIPO IV, *Proceedings of the 25th Workshop of the UK Planning and Scheduling SIG (PLANSIG-06)*, pp. 55–65. 48, 100

McCluskey, T. L. and Simpson, R. M. (2004). Knowledge Formulation for AI Planning, Proceedings of 4th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004) Whittlebury Hall, Northamptonshire, UK, 2004. Published by Springer in the LNAI series. xii, 19, 50

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

URL: *http://eprints.hud.ac.uk/7859/* 27

McDermott, D. (1997). The Classical Planning Problem Specification Language - A Manual, *Technical report*, Yale University. 8

McDermott, J. (1998). 1998 AIPS planning competiton, ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html. 3, 38, 83

Murata, T. (1989). Petri nets: Properties, analysis and applications, *Proceedings of the IEEE* **77**(4): 541 –580. 45

Muscettola, N., Nayak, P. P., Pell, B. and Williams, B. C. (1998). Remote Agent: To Boldly Go Where No AI System Has Gone Before, *Artificial Intelligence* **103(1-2)**: 5–48. 4, 25

Nakhost, H., Müller, M., Valenzano, R. and Xie, F. (2011). Arvand: the art of random walks, *García-Olaya et al.(2011)* pp. 15–16. 14

Nau, D. S. (2007). Current trends in automated planning, *AI Magazine* **28**(4): 43. **URL:** *http://www.aaai.org/ojs/index.php/aimagazine/article/view/2067* 3, 14

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D. and Yaman, F. (2003). Shop2: An htn planning system, *J. Artif. Intell. Res. (JAIR)* **20**: 379–404. 14

Naveed, M., Kitchin, D. E., Crampton, A., Chrpa, L. and Gregory, P. (2012). A monte-carlo path planner for dynamic and partially observable environments, *CIG*, pp. 211–218. 1

Newell, A. and Simon, H. A. (1963). GPS: A Program that Simulates Human Thought, *in* E. A. Feigenbaum and J. Feldman (ed.), *Computers and Thought*, R. Oldenbourg KG. 12

Owens, N., Armstrong, A., Sullivan, P., Mitchell, C., Newton, D., Brewster, R. and Trego, T. (2000). Traffic Incident Management Handbook, *Technical Report Office of Travel Management*, Federal Highway Administration. 66, 68

Parkinson, S., Longstaff, A. P., Crampton, A. and Gregory, P. (2011). The application

of automated planning to machine tool calibration, *In Proceeding of 22nd International Conference on Automated Planning and Scheduling(ICAPS11)*. 1

Pearson, D. J. and Laird, J. E. (1996). Toward incremental knowledge correction for agents in complex environments, *in* S. Muggleton, D. Michie and K. Furukawa (eds), *Machine Intelligence*, Vol. 15, Oxford University Press. 12

Riabov, A. and Liu, Z. (2006). Scalable planning for distributed stream processing systems, *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*, Cumbria, UK, pp. 31 – 41. 26

Richter, S. and Westphal, M. (2008). The lama planner using landmark counting in heuristic search, *Proceedings of IPC*. 14

Richter, S., Westphal, M. and Helmert, M. (2011). Lama 2008 and 2011, *The 2011 International Planning Competition* p. 50. 14

Roberts, M., Howe, A. and Flom, L. (2007). Learned models of performance for many planners, *ICAPS 2007 Workshop AI Planning and Learning*, pp. 36–40. 123

Rumbaugh, J., Jacobson, I. and Booch, G. (2004). *Unified Modeling Language Reference Manual, The*, Pearson Higher Education. 43, 44

Sacerdoti, E. (1974). Planning in a Hierarchy of Abstraction Spaces, *Artificial Intelligence* **5**. 12

Sacerdoti, E. (1975). The Nonlinear Nature of Plans, *Proc. IJCAI*. 13

Shah, M., Chrpa, L., Gregory, P., McCluskey, T. and Jimoh, F. (2012). Ocl plus:processes and events in object-centred planning, *Frontiers in Artificial Intelligence and Applications*, Vol. 241 of *STAIRS 2012 - Proceedings of the 6th Starting*

AI Researchers' Symposium, IOS Press, pp. 282–293.

URL: *http://eprints.hud.ac.uk/14667/* 27

Simpson, R., Kitchin, D. E. and McCluskey, T. (2007). Planning domain definition using gipo, *Knowledge Engineering Review* **22**(2): 117–134. 4, 5, 19, 29, 48, 89, 100, 122

Simpson, R. M. (2005). GIPO: graphical interface for planning with objects, *Proceedings of the International Competition on Knowledge Engineering in Planning and Scheduling, Monterey, California*. 137

Simpson, R. M., McCluskey, T. L. and Liu, D. (2000). OCLGraph : Exploiting Object Structure in a Plan Graph Algorithm, *Proceedings of the Workshop on New Results in Planning, Scheduling and Design (PuK2000) at ECAI 2000*. 27

Simpson, R. M., McCluskey, T. L., Zhao, W., Aylett, R. S. and Doniat, C. (2001). GIPO: An Integrated Graphical Tool to support Knowledge Engineering in AI Planning, *Proceedings of the 6th European Conference on Planning (ECP-01)*, Springer-Verlag, pp. 544 – 552. 4, 27

Smith, D. E., Frank, J. and Cushing, W. (2008). The anml language, *Proceedings of ICAPS-08* . 25, 51

Sommerville, I. (2004). *Software Engineering: Seventh Edition*, Pearson Education.

URL: *http://books.google.co.uk/books?id=PqsWaBkFh1wC* 21

Tate, A. (1977). Generating Project Networks, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. 2, 13, 18, 39

Tate, A., Drabble, B. and Dalton, J. (1996). O-Plan: a Knowledged-Based Planner and its Application to Logistics, AIAI, University of Edinburgh. 1, 40

Tate, A., Drabble, B. and Kirby, R. (1994). O-Plan2: an Open Architecture for Command, Planning and Control, *in* M. Fox and M. Zweben (eds), *Intelligent Scheduling*, Morgan Kaufmann. 18, 29, 39

Tate, A., Polyak, S. T. and Jarvis, P. (1998). TF Method: An Initial Framework for Modelling and Analysing Planning Domains, *Technical report*, University of Edinburgh. 39

Tomas Plch, Miroslav Chomut, C. B. R. B. (2012). Inspect, edit and debug pddl documents: Simply and efficiently with pddl studio, *ICAPS12 System Demonstration* p. 4. 54

Vaquero, T. S., Romero, V., Tonidandel, F. and Silva, J. R. (2007). itSIMPLE2.0: An integrated tool for designing planning domains, *Proceedings of the 17th International Conference on Automated Planning & Scheduling (ICAPS-07)*, AAAI Press, pp. 336–343. 4, 40, 44, 89

Vaquero, T. S., Silva, J. R. and Beck, J. C. (2010). Analyzing plans and planners in itSIMPLE3.1, *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop – The 20th International Conference on Automated Planning & Scheduling (ICAPS-10)*. 4

Vaquero, T. S., Silva, J. R. and Beck, J. C. (2011a). Acquisition and re-use of plan evaluation rationales on post-design, *KEPS 2011* p. 15. 5

Vaquero, T. S., Silva, J. R. and Beck, J. C. (2011b). A brief review of tools and methods

for knowledge engineering for planning & scheduling, *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop – The 21th International Conference on Automated Planning & Scheduling (ICAPS-11)*. 4, 21

Vaquero, T. S., Silva, J. R. and Beck, J. C. (2011c). A conceptual framework for post-design analysis in ai planning applications, *KEPS 2011* p. 109. 5

Vaquero, T. S., Silva, J. R., Tonidandel, F., Ferreira, M. and Beck, J. C. (2009). From requirements and analysis to PDDL in itSIMPLE3.0, *Working notes of the 19th International Conference on Automated Planning & Scheduling (ICAPS 2009) – The 3rd International Competition on on Knowledge Engineering for Planning and Scheduling (ICKEPS)*. 4, 42

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, *System Demonstration – Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)*. 4, 5, 29, 40, 89, 95

Wickler, G. (2011). Using planning domain features to facilitate knowledge engineering, *KEPS 2011* p. 39. 154

Wilkes, D. E. and Myers, K. L. (1995). A common knowledge representation for plan generation and reactive execution, *Journal of Logic and Computation* **5**(6): 731–761. 3

Wilkins, D. (1988). *Practical Planning: Extending the Classical AI Paradigm*, Addison-Wesley. 23

Wilkins, D. (1999). Using the SIPE-2 Planning System: A Manual for SIPE-2, Version5.0, SRI International, Artificial Intelligence Center. 3

Wilkins, D. and desJardins, M. (2000). A Call for Knowledge-based Planning, *Proceedings of the 2nd NASA International Workshop on Planning and Scheduling for Space*, p. 187. 2

Wilkins, D. and Myers, K. (1994). A Common Knowledge Representation for Plan Generation and Reactive Execution, *Journal of Logic and Computation* . 29

Wu, K., Yang, Q. and Jiang, Y. (2005). Arms: Action-relation modelling system for learning acquisition models, *Proceedings of the First International Competition on Knowledge Engineering for AI Planning*, Monterey, California, USA. 154

Xing, Z., Chen, Y. and Zhang, W. (2006). Maxplan: Optimal planning by decomposed satisfiability and backward reduction, *Proceedings of the Fifteenth International Planning Competition, International Conference on Automated Planning and Scheduling*, pp. 53–56. 14