

A knowledge engineering methodology for rapid prototyping of planning applications

Luis Castillo and Juan Fdez-Olivares and Antonio González

and Gonzalo Milla and David Prior and Lluvia Morales and José Figueroa

Departamento de Ciencias de la Computación e I.A.

ETSI Informática y Telecomunicaciones

Universidad de Granada.

Víctor Pérez-Villar

Boeing Research and Technology Europe

Abstract

When deploying planning applications as part of greater intelligent systems, either of small size or large-scale systems, there is an underlying knowledge acquisition, engineering and representation process that has to be undertaken with regards to the successful integration of a planning engine into the whole architecture of the system. In this paper we propose a methodology, based on the use of the ontology editor Protégé, that enables both a fast and easy knowledge acquisition, engineering an representation of planning domains and it also enables an easy knowledge sharing between the planning engine and either other subsystems or even end users. This is a preliminary work that is being used within a research contract with Boeing Research and Technology Europe as part of the Project Integra founded by CDTI in the framework of the Spanish CENIT Program.

Introduction

Knowledge acquisition and representation is a key issue in the development of planning applications either for small domains or for large-scale deployments. In most cases, planning engines are just one more part of the whole architecture and the integration of planning algorithms cannot ignore this because the domain knowledge that they use to work is also shared with other systems or even human operators. Therefore, this domain knowledge, that in the case of planning domains usually has the form of predicate logic, has to be interfaced with these other components of the whole picture, either humans or programs. In addition to this, planning domain knowledge is not always explicitly stated since the beginning. In some cases, there is a weak formalization of the knowledge, usually in terms of databases, that is, very often, insufficiently detailed. In other cases there is no such initial representation and a brand new one has to be designed from scratch. These are the coordinates of the main contribution of this paper: enabling an easy *knowledge acquisition* stage of planning domain knowledge and enabling a good level of *knowledge sharing* either with other processes or human operators.

In order to achieve this two-folded goal a methodology based on the use of ontologies, and in particular the Protégé ontology editor framework (National Library of Medicine

2009), is proposed to guide the acquisition, engineering and exchange of planning domain knowledge¹.

Ontologies have been widely used in many general AI approaches and applications (Uschold and Gruninger 1996), and particularly as a mean to support Knowledge Engineering processes (Schreiber, Crubézy, and Musen 2000). They are related to the representation of the entities of a given domain, their properties and their relations. In particular, in the field of intelligent planning ((Valente et al. 1999; Fdez-Olivares et al. 2006), ontologies have been used primarily to represent the part of the domain knowledge related to the *objects model* (entities, their properties and relationships) as well as to the *initial state* like the set of objects instances present in the problem and their initial properties. However there are other approaches that have also used ontologies for representing plans and tasks in a holistic view (Gil and Blythe 2000) or high level representations that might be used to build domain actions and tasks in an HTN framework like in web service composition (Fdez-Olivares et al. 2007) or in a distance learning framework (Castillo et al. 2009; Kontopoulos et al. 2008).

In the case of this paper we will focus on how, once the knowledge relative to the planning objects model and the planning problem has been modeled within Protégé, this knowledge model can be easily and quickly translated into a representation understandable by a planner. Concretely, we suggest the automatic generation and representation of all the sections of the problem and domain files in PDDL 2.1 (Long and Fox 2003) except those related to durative actions for non-hierarchical planning engines and, those related to compound tasks, in the case of HTN planning engines. Hence, the problem file in PDDL 2.1 syntax might be fully generated by an automatic translation program and, regarding the PDDL domain file, the sections :types, :predicates, :functions and :constants might also be automatically generated, that is, the whole domain except those sections related to actions.

When a brand new planning application is being built, the planning domain knowledge is under constant evolution and

¹This is a preliminary work that is being used in a research contract with Boeing Research and Technology Europe as part of the Project Integra founded by CDTI (Spanish Center for the Industrial Technological Development) in the framework of the Spanish CENIT R&D Program.

changes. The methodology presented in this paper allows both, an easier exchange of knowledge with other subsystems or human operators, and a quick and robust regeneration of most PDDL sentences both in the problem and in the domain files, leading to a faster and reliable evolution of the planning application.

Next sections frames this problem in more detail and explains why this approach is useful.

The life-cycle of a brand new planning application

The building of a planning application from scratch is a handcrafted work nowadays. Planning engineers have to study the problem at hand and build a knowledge model suitable for representing all the details needed for the planning engine to work and even some additional details related to administrative, legal or control issues. If some previous database is available it may help a little but since these data repositories were not designed to be used by a planning algorithm, they usually lack of the desired level of detail or precision and many times they need to be re-worked anyway. As the project goes on, and following a typical, spiral-based knowledge acquisition scheme based on interviews and prototypes, new capabilities of the planning engine are incrementally agreed between end users and planning engineers, in the form of new types of objects to be considered, or new actions to be taken or new details to be included into already existing actions. This leads to an enormous amount of planning engineering work in the form of representing new PDDL predicates, initial states, objects, properties and so on. And last, but no least, potential syntactic mistakes during hand-coding PDDL or any other suitable language are easily avoided with an automation of this process. The more complex the application scenario is, the greater this effort is too, increasing the possibility of introducing errors, putting in risk the reliability of the system and increasing the cost of the application.

This knowledge acquisition effort can also be supported by existing methodologies, like CommonKADS, so that it may be carried out more formally.

The CommonKADS underlying timeline

Planning engineers have the choice to follow CommonKADS methodology (Schreiber et al. 1999) to give a formal envelope to this knowledge engineering effort. Previous work (Fdez-Olivares et al. 2006; Kingston, Shadbolt, and Tate 1996) show that, since a planning application is also a knowledge-based system, this methodology can be used as the underlying timeline for the design and representation of planning domain knowledge (see Figure 1).

This methodology establishes three main stages devoted to the (1) functional requirements analysis, (2) knowledge requirements analysis and (3) design and building of a knowledge-based system (KBS). The first stage is focused on the analysis of the context where a KBS (in our case a planning application) is intended to operate. Three different models are provided to this stage:

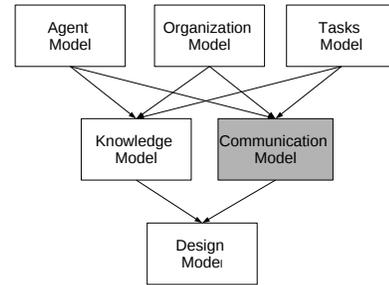


Figure 1: CommonKADS models for knowledge acquisition and engineering

- **Organization Model.** This is the *physical* set of objects that take part in the domain application: organizational units and their (usually hierarchical) structure, resources, facilities, infrastructures, etc. It also contains their properties and relations.
- **Agent Model.** It is devoted to model the executors of tasks (humans, information systems or any other organizational entity capable of carrying out a task). It also describes some *non-physical* features related to organizational knowledge like command and control constraints, authorities, roles and communication channels, legal and/or administrative issues, etc.
- **(Business) Task Model** It is used to analyze and define the workflow processes (either knowledge-intensive or not) in which the entities of the above models take part (tasks in this model are sub-parts of the whole *business process* that describes the operation of the system).

The second stage is supported by the following two models:

- **Knowledge Model.** It specifies the knowledge and reasoning requirements of those knowledge-intensive information-processing tasks selected from the Task Model in order to identify the expertise available in the problem. The model is subdivided in three *knowledge categories*: *domain knowledge* (comparable to an object model), *inference knowledge* (inference steps of the system, like heuristics or refined ad-hoc rules, seen as the building blocks of the reasoning machinery) and *task knowledge* (goals and how they are realized through decompositions into sub-tasks and (ultimately) inferences).
- **Communication Model.** It is related to the transfer of information between the agents, specifying interfaces and interactions of the system with other agents, either human or other software systems.

Finally, the Design Model is the final stage of the methodology devoted to describe the global architectural blocks needed to accomplish the functionalities detected in previous stages.

In a planning setting, the agent and organization model might gather all the knowledge available about the domain objects that take part in the system, the relations between

them and the existing constraints, that finally is represented as the *domain knowledge* category of the Knowledge Model. The task model may be used to detect those knowledge-intensive processes that might be better supported by planning techniques. The knowledge model (apart from the object model) may contain all inferencing mechanisms of the problem, that is, abductive or deductive inference processes needed in the planning process or even the knowledge relative to actions and to compound tasks, decomposition schemes, in the case of HTN planning. Communication model may be used to establish a way to define mixed-initiative or collaborative problem solving strategies, though it does seem to provide much better support in a planning application for the stages of plan execution and monitoring, that is out of the scope of this paper. The design model might be understood as the representation of all the above knowledge sources into the final representation, in our case, the PDDL domain and problem files ready to be used by a planning engine.

Let us see how the ontology editor Protégé may be used to cover the agent, organization and knowledge models. It is important to remark that the ontology finally obtained in this process not only covers all these stages of planning domain knowledge engineering, but also achieves the goals of knowledge sharing and exchange between different processes and human operators. Finally, although there are some works in the literature that also focus on the representation of both, the (business)Task Model and *task knowledge* category of the Knowledge Model in the same ontology (Gil and Blythe 2000; Fdez-Olivares et al. 2007), these aspects of a planning domain are currently under study and it is also left out of the scope of the paper.

The use of the Protégé ontology editor

This section shows the advantages of using the Protégé Frames ontology editor for representing planning domain knowledge following the CommonKADS timeline for the agents model, the organization model, and the design model.

Firstly, it allows to easily represent planning domain knowledge (except that related to actions) in a well known framework with a graphical interface, focusing on the relevant aspects of this knowledge: objects, properties and relations and ignoring the details of such representation in predicate logic. This easily allows the successive revisions to be carried out on this domain knowledge through the life cycle of a planning application. This ease of editing planning domain knowledge would be nothing if this knowledge, represented in the standard Protégé Frames form (National Library of Medicine 2009), cannot be made readable by any state of the art planner. However, as the next section shows, the knowledge represented in Protégé Frames form can be translated into usual PDDL syntax with an automated procedure and all the sections of PDDL 2.1 domain and problem file can be automatically generated. This translation process is a great advantage since it immediately provides valid PDDL files after each revision of the ontology.

Secondly, the knowledge in the Protégé ontology can also be easily shared and exchanged both with other components of the application like web applications (Fdez-Olivares et al.

2006), GIS systems like Google Maps, or other databases and also with human operators. This allows end users to modify planning domain knowledge from easy-to-use web forms without having to know about ontologies or predicate logic or artificial intelligence in general.

In order to provide these advantages, the following steps have been given. On the one hand, the Protégé ontology has to be backed up on a back-end database in MySQL (see Figure 2). This is automatically done through the Protégé GUI. On the other hand, a web service named Ontoserver has been designed and implemented over that back-end providing simultaneous access through the web to the ontology and allowing most kind of queries to the ontology such as browsing the hierarchy of classes, querying instances, following links of properties, etc.

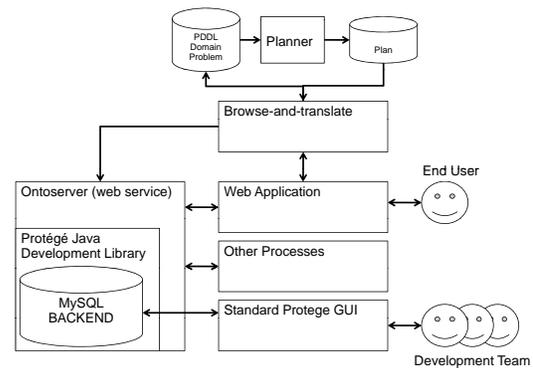


Figure 2: Architecture of a possible implementation of the methodology. The original ontology has been backed up in a MySQL database. This remote database may be used directly from the original Protégé GUI like a regular ontology by the development team to modify the ontology. Over this back-end, and thanks to the development libraries of Protégé, a web service named Ontoserver has been implemented. The role of this web service is allowing other processes to access the ontology like web applications or the translation procedures that translate the knowledge in the ontology into valid PDDL 2.1 domain and problem files.

Automating planning domain and problem file generation during the planning application life-cycle

Thanks to the use of the Ontoserver process, the ontology can be fully browsed and all the available knowledge can be easily translated into PDDL form. Prior to explain these browse-and-translate algorithms some initial assumptions have to be made.

- The ontology is composed of a tree (or graph) of available classes
- Each class has a set of slots that define the set of properties of every instance of each class. The slots of a given class

may be inherited slots, coming from superclasses of the class, and direct slots, that are not inherited.

- All the slots of any class in the ontology may be of any of the following types:
 - STRING. Its value is a sequence of characters.
 - SYMBOL. Its value is a symbol out of a set of possible values.
 - INTEGER. An integer number.
 - FLOAT. A floating point number.
 - BOOLEAN. True or false slots.
 - INSTANCE. A reference to an existing instance of an allowed class or classes.

Generating PDDL domain sections All the sections of a PDDL domain file are generated, except those related to actions schemes. One of the easiest PDDL sections to be generated is the :TYPES section. It is generated just by browsing the hierarchy of available classes in the ontology (Figure 3). Each class in the ontology is translated into a valid PDDL type.

```

1. Let  $\mathcal{C}$  be the root class of the ontology,  $\mathcal{V} = \emptyset$  the list of already visited classes of the ontology,  $\mathcal{Q}$  the queue of classes to be processed,  $\mathcal{Q} = \{\mathcal{C}\}$ 
2. Generate the first row, write "( :TYPES  $\mathcal{C}$  - object"
3. while  $\mathcal{Q} \neq \emptyset$ 
  (a) Let  $q = extract(\mathcal{Q})$ 
  (b) if  $q \notin \mathcal{V}$  then
    i. let  $\{q_1, q_2, \dots, q_n\}$  be the set of subclasses of class  $q$ 
    ii. write " $q_1$   $q_2$  ...  $q_n$  -  $q$ "
    iii. Append  $\{q_1, q_2, \dots, q_n\}$  to  $\mathcal{Q}$ 
  (c) Append  $q$  to  $\mathcal{V}$ 
4. write ")"

```

Figure 3: Algorithm Types Generation (TG). Generates the PDDL section :types from the hierarchy of classes of the ontology.

Another very useful section is the :CONSTANTS section. This section must contain all valid symbols to be shared between any problem instance of the planning domain. In this case, all the possible values of a given enumerated slot are translated into valid constants.

The PDDL section :PREDICATES and :FUNCTIONS are extremely important since they are the main tool to encode the most important knowledge to be considered by PDDL actions and states. Every instance of the ontology has a set of slots. Every slot represents a property (STRING, SYMBOL, INTEGER, BOOLEAN, FLOAT) and has a value associated to it. Non-numerical slots (Figure 6) are treated as regular predicates following the scheme (<slot-name> <instance-name> <slot-value>). BOOLEAN slots are treated in a special way following the closed world assumption, if they are true, then they are translated as (<slot-name> <instance-name>). Otherwise they are not translated. Numerical slots

```

1. Let  $\mathcal{C}$  be the root class of the ontology,  $\mathcal{V} = \emptyset$  the list of already visited slots of the ontology
2. Generate the first row, write "( :CONSTANTS "
3. for each subclass  $c$  of  $\mathcal{C}$ 
  (a) for each slot  $s$  in  $c$ 
  (b) if  $s \notin \mathcal{V}$  and  $s$  is a direct slot of  $c$  and  $s$  is of type SYMBOL then
    i. for each possible value  $x$  of the slot  $s$  write " $x$ "
  (c) Append  $s$  to  $\mathcal{V}$ 
4. write ")"

```

Figure 4: Algorithm Constants Generation (CG). Generates the PDDL section :constants from the set of available enumerated SYMBOLS of the slots of the ontology. Slots of type STRING can also be generated translating each string into a symbol like for example translating the string "This is a test" into this symbol This_is_a_test

(Figure 7) are translated as functions following the scheme (= (<slot-name> <instance-name> <numeric-value>). And finally, slots of type INSTANCE (Figure 6), that represent binary relations between instances (or classes), are translated following the scheme (<slot-name> <source-instance-name> <target-instance-name>). See Figure 5 for a simple example.

Instance	INSTANCE_054
Code	AF_56_OO
Height	100 m
Is Part of	INSTANCE_010
Available	true

```

(code INSTANCE_054 AF_56_00)
(= (height INSTANCE_054) 100)
(is-part-of INSTANCE_054 INSTANCE_010)
(available INSTANCE_054)

```

Figure 5: Basic translation scheme for each instance.

Generating PDDL problem file Having seen the translation and encoding of all the sections of the domain file, it is very easy to figure out that the problem file follows the same scheme of the browse-and-translate procedures seen before (Figure 8). Every instance of the ontology is translated into a PDDL typed object in the problem. The type of the object is the class of the ontology it belongs to. And the initial state is a simple dump of all the slots of all the objects present in the ontology. The goal of the problem may be specified as a parameter of the translation algorithm. In the case of non-HTN planners, it must consist of a sequence of literals to be made true. In the case of HTN planners it must consist of a sequence of tasks to be decomposed.

In summary, these sections have shown how PDDL problem and domain files can be easily generated from a Protégé ontology, allowing an easy adaptation process to the evolu-

1. Let \mathcal{C} be the root class of the ontology, $\mathcal{V} = \emptyset$ the list of already visited slots of the ontology
2. Generate the first row, write "(:PREDICATES"
3. for each subclass c of \mathcal{C}
 - (a) for each slot s in c
 - (b) if $s \notin \mathcal{V}$ and s is a direct slot of c then
 - i. let $D = \{d_1, d_2, \dots, d_m\}$ the domain of slot s , that is the class or the set of classes to which s belongs.
 - ii. if s is of type BOOLEAN then
 - A. if $|D| = 1$ then write "(s ?x - d_1)"
 - else write "(s ?x - (either D))"
 - iii. if s is of type SYMBOL or STRING then
 - A. if $|D| = 1$ then write "(s ?x - d_1 ?y - object)"
 - else write "(s ?x - (either D) ?y - object)"
 - iv. if s is of type INSTANCE then
 - A. let $I = \{i_1, i_2, \dots, i_t\}$ the set of ID's of classes allowed as instances associated to the slot s
 - B. if $|I| = 1$ then
 - if $|D| = 1$ then write "(s ?x - d_1 ?y - i_1)"
 - else write "(s ?x - (either D) ?y - i_1)"
 - C. else
 - if $|D| = 1$ then write "(s ?x - d_1 ?y - (either I))"
 - else write "(s ?x - (either D) ?y - (either I))"
 - (c) Append s to \mathcal{V}
 4. write ")"

Figure 6: Algorithm Predicates Generation (PG). Generates the PDDL section `:predicates` from the set of available slots that are not INTEGER or FLOAT.

tion of the planning domain knowledge during the life-cycle of a planning application. Another relevant aspect of building planning application is the exchange of knowledge between the planner and other components of the whole architecture or even human operators who cannot be supposed to handle concepts of artificial intelligence or predicate logic.

Knowledge sharing

In this sense, having plain PDDL domain a problem files as the source files for the planning knowledge is a huge drawback to enable this exchange of knowledge between all the agents involved in the architecture shown in Figure 2. However, given the infrastructure commented in previous sections, this task is much easier. It's time to recall that the Ontoserver process is a web service that allows to browse the ontology, its classes and instances through a clearly defined API (Advanced Programming Interface). This allows other systems to remotely access the ontology in a interoperable manner and also to build interfaces so that human operators can also access the ontology through any web browser.

The project INTEGRA

In the case of this paper, an intelligent decision support system is being built under a research contract with Boeing Research and Technology Europe within the Project INTEGRA

1. Let \mathcal{C} be the root class of the ontology, $\mathcal{V} = \emptyset$ the list of already visited slots of the ontology
2. Generate the first row, write "(:FUNCTIONS"
3. for each subclass c of \mathcal{C}
 - (a) for each slot s in c
 - (b) if $s \notin \mathcal{V}$ and s is a direct slot of c then
 - i. let $D = \{d_1, d_2, \dots, d_m\}$ the domain of slot s , that is the class or the set of classes to which s belongs.
 - ii. if s is of type INTEGER or FLOAT then
 - A. if $|D| = 1$ then write "(s ?x - d_1)"
 - else write "(s ?x - (either D))"
 - (c) Append s to \mathcal{V}
4. write ")"

Figure 7: Algorithm Fluents Generation (FG). Generates the PDDL section `:functions` from the set of available slots that are of type INTEGER or FLOAT.

founded by CDTI in the framework of the Spanish CENIT Program. Project INTEGRA is devoted to explore the use of a broad set of new technologies to the management and control of migration flows. In particular, planning technology is being used to support the decisions of Command and Control members during the occurrence of a migration incident (arrival of illegal migrants, illegal commercial traffic, etc) or even in potential scenarios in which a future incident is predicted by a data mining module of the project.

Related work

One of the most close tools to the one here presented is itSIMPLE (Vaquero et al. 2009). As our ontology-based methodology, the itSIMPLE environment aims to help designers to overcome the problems encountered during life cycle of planning application projects, mainly at the requirements specification, modeling and analysis phases. itSIMPLE is designed to permit users to have a disciplined design process to create knowledge intensive models for several planning domains. itSimple is based on a translation from a domain model expressed in UML to PDDL. This tool is aimed to bridge the gap between traditional Software Engineering methodologies and Knowledge Engineering for Planning and Scheduling. However, unless the methodology here presented, it does not build on more appropriate methodologies for KB Systems like CommonKADS, nor is based on an ontology in order to represent the various models of CommonKADS. This last argument is a very important one, since the main advantage of using an ontology-based methodology is that crucial issues for intelligent systems integration like knowledge sharing and exchange can be faced with little effort, and therefore easily and quickly developed. Nevertheless, itSimple does not addresses these issues since its internal representation language (XPDDL, a XML-extension of PDDL) is not designed under interoperability criteria.

```

1. write "(OBJECTS " ;; :OBJECTS section
2. Let  $\mathcal{C}$  be the root class of the ontology
3. for each subclass  $c$  of  $\mathcal{C}$ 
  (a) for each instance  $i$  of  $c$ 
    i. write " $i - c$ "
4. write ")"
5. write "(INIT " ;; :INIT section
6. Let  $\mathcal{C}$  be the root class of the ontology
7. for each subclass  $c$  of  $\mathcal{C}$ 
  (a) for each instance  $i$  of  $c$ 
    i. for each slot  $s$  of  $i$ 
      A. if  $s$  is of type SYMBOL or STRING
        then write " $(s i \text{ value}(s))$ "
      B. if  $s$  is of type BOOLEAN and  $\text{value}(s) = \text{TRUE}$ 
        then write " $(s i)$ "
      C. if  $s$  is of type INSTANCE
        then write " $(s i \text{ instance}(s))$ "
      D. if  $s$  is of type INTEGER or FLOAT
        then write " $(= (s i) \text{ value}(s))$ "
8. write ")"

```

Figure 8: Algorithm Problem Generation (PrG). Generates a full PDDL problem.

Concluding remarks

This paper has presented an approach to support planning domain knowledge acquisition and engineering for the integration of planning engines into greater applications. It brings planning engineering a little closer to end user knowledge and viceversa, boosting and improving the communication between planning engineers and end users, which always implies savings on time and also money whenever it is involved. Two are the main obstacles to be overcome in this task of integrating a planning engine into a planning application. On the one hand, tools have to be provided to encode planning domain knowledge as easier as possible and to adapt this knowledge as the application is enhanced or new details are considered, following the life-cycle of a planning application. On the other hand, this planning domain knowledge must not be owned exclusively by the planning engine, instead, it must be shared with other subsystems of the whole architecture and it must be exchanged with ease with human operators in charge of handling the system. The paper has shown how the use of the Protégé ontology editor framework together with an underlying CommonKADS methodology can provide the means to overcome these two obstacles with regards to an easier and faster prototyping of planning applications.

References

- Castillo, L.; Morales, L.; González-Ferrer, A.; Fdez-Olivares, J.; Borrajo, D.; and Onaindía, E. 2009. Automatic generation of temporal planning domains for e-learning problems. *Journal of Scheduling*.
- Fdez-Olivares, J.; Castillo, L.; García-Pérez, O.; and Palao, F. 2006. Bringing users and planning technology together. experiences in SIADEX. In *Sixteenth International Conference on Automated Planning and Scheduling, ICAPS*. Awarded as the Best Application Paper of this edition.
- Fdez-Olivares, J.; Garzón, T.; Castillo, L.; García-Pérez, O.; and Palao, F. 2007. A Middleware for the automated composition and invocation of semantic web services based on HTN planning techniques. In *Current topics in Artificial Intelligence, CAEPIA07, Lecture Notes on AI 4788*. Springer.
- Gil, Y., and Blythe, J. 2000. PLANET: A shareable and reusable ontology for representing plans. In *AAAI 2000 workshop on representational issues for real-world planning systems*.
- Kingston, J.; Shadbolt, N.; and Tate, A. 1996. CommonKADS models for knowledge based planning. *TECHNICAL REPORT-UNIVERSITY OF EDINBURGH ARTIFICIAL INTELLIGENCE APPLICATIONS INSTITUTE AIAI TR*.
- Kontopoulos, E.; Vrakas, D.; Kokkoras, F.; Bassiliades, N.; and Vlahavas, I. 2008. An ontology-based planning system for e-course generation. *Expert Systems with Applications* 35(1-2):398–406.
- Long, D., and Fox, M. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61–124.
- National Library of Medicine. 2009. <http://protege.stanford.edu/>. Version 3.4.1.
- Schreiber, G.; Akkermans, H.; Anjewierden, A.; de Hoog, R.; Shadbolt, N.; de Velde, W. V.; and Wielinga, B. 1999. *Knowledge Engineering and Management – The CommonKADS Methodology*. The MIT Press.
- Schreiber, G.; Crubézy, M.; and Musen, M. A. 2000. A case study in using protégé-2000 as a tool for commonkads. In *EKAW '00: Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, 33–48. London, UK: Springer-Verlag.
- Uschold, M., and Gruninger, M. 1996. Ontologies: Principles, methods and applications. *Knowledge Engineering Review* 11(2).
- Valente, A.; Russ, T.; MacGregor, R.; and Swartout, W. 1999. Building and (re) using an ontology of air campaign planning. *IEEE Intelligent Systems and their Applications* 14(1):27–36.
- Vaquero, T. S.; Silva, J. R.; Ferreira, M.; Tonidandel, F.; and Beck, J. C. 2009. From requirements and analysis to pddl in itsimple3.0. In *ICKEPS'09: Proceedings of the 3rd. International Competition on Knowledge Engineering for Planning and Scheduling*, 54–61. Thessaloniki, Greece: ICAPS 2009.