Using Ontologies as Semantic Representations of Hierarchical Task Network Planning Domains

Artur Freitas, Daniela Schmidt,
Felipe Meneguzzi, Renata Vieira and Rafael H. Bordini

Faculty of Informatics, PUCRS - Brazil
{artur.freitas,daniela.schmidt}@acad.pucrs.br,
{felipe.meneguzzi,renata.vieira,rafael.bordini}@pucrs.br

Abstract. Integrating knowledge representation approaches, such as ontologies, with the field of automated planning is still an open research challenge. To explore this issue, we present a semantic model to address the knowledge representation of planning domains. More specifically, we show an ontological approach to represent HTN (Hierarchical Task Network) in OWL (Web Ontology Language) ontologies. We explain our planning ontology, demonstrate one instantiation to exemplify its use and propose algorithms to convert specifications from OWL to HTN, and vice-versa. We also discuss some future directions towards the integration of planning formalisms and semantic representations.

1 Introduction

Knowledge representation approaches using ontologies are being studied as promising techniques to enable semantic reasoning, knowledge reuse, interoperability, and so on. Yet, the use of ontologies integrated with planning formalisms is a research path that is currently at its initial steps. To address this issue, this paper proposes a semantic model to represent planning domains. The proposed ontology was encoded in OWL (Web Ontology Language) [1] to model planning problems based on the HTN (Hierarchical Task Network) paradigm [2]. This conceptualisation was instantiated in ProtÃľgÃľ[1] to demonstrate how planning domains can be modelled in ontologies. We also present a bidirectional mapping among HTN domain formalisations and their respective OWL ontology representations. We show the algorithms to automatically translate from OWL to HTN specifications (and vice-versa) that were implemented in Java using the OWL API [3]. Therefore, planning domains instantiated in the ontology can be automatically translated to the HTN input specification (and the other way around) using these two aforementioned methods. This work aligns the fields of knowledge representation and reasoning with the domain of automated planning, and this opens the path to interesting research directions that are still in their initial steps in the scientific community. For instance, our approach enables to deriving planning domain models from existing ontological knowledge, and also to convert again from the ontology to a planning domain.

In other words, we investigate the integration of ontologies and the HTN formalism in order to explore semantic representations of planning domains. HTN planning is like

[1] http://protege.stanford.edu/

classical AI planning in that each state of the world is represented by a set of atoms, and each action corresponds to a deterministic state transition [4]. However, HTN planners differ from classical AI planners in what they plan for, and how they plan for it. In classical planning, the main aim of the planning task is to attain a goal-state, which is usually specified in terms of a number of desired properties of the world. In this context, JSHOP (Java Simple Hierarchical Ordered Planner) is a domain-independent automated-planning system based on *ordered task decomposition*, which is a type of HTN planning. On the other hand, ontology is a knowledge representation structure composed of concepts, properties, instances, relationships, and axioms, which is defined as an "explicit specification of a conceptualisation" [5]. Thus, our goal with this research is to explore and demonstrate the utilisation of ontologies more expressively in automated planning.

This paper is organised as follows. Next section provides a background on ontologies. A section of related work is presented afterwards. Then, a section explaining our conceptualisation (TBox, *i.e.*, Terminological Box) in OWL is presented. This conceptualisation is composed of classes and properties to represent the HTN domain. Next, we show an instantiation (ABox, *i.e.*, Assertion Box) of this TBox in order to demonstrate how to use the proposed ontology to model a corresponding HTN specification. Also, we present the two algorithms coded in Java with the OWL API [3] to, respectively convert from OWL to HTN, and vice-versa. Then, we conclude this research direction and point out other possible investigations towards the integration of ontology and planning.

2 Background

Ontology was born as a philosophical study of reality aiming at defining which things exists in reality and what we can say about them. Researchers in Artificial Intelligence and Computer Science define ontology as an "explicit specification of a conceptualisation" [5]. In this context, a conceptualisation means an abstract model of some aspect of the world which defines the properties of important concepts and relationships. From this definition, we can observe that an ontology is a knowledge representation structure composed of concepts, properties, individuals, relationships and axioms [6], as described in sequence. A **concept** is an abstract group, set, class or collection of objects that share common properties. This component is represented in hierarchical graphs, such that it looks similar to object-oriented systems. A **property** is used to express relationships between concepts in a given domain. More specifically, it describes the relationship between the first concept (*i.e.*, the domain), and the second, which represents that property range. For example, "study" could be represented as a relationship between the concept "person" (as the property domain) and "university" or "college" (as range). An **individual** is the "ground-level" component of an ontology which represents a specific element of a concept or class. Individuals are also called instances, objects or facts. A **relationship** is an instance of a property, which relates two individuals: one as domain of the relationship, and another as its range. It is important that those individuals obey the constraints represented in the property specification in order to guarantee the consistency of the ontology instantiation. An **axiom** is used to impose constraints on the values of classes or individuals, so axioms are generally expressed using logic-based languages, such as first-order logic. Axioms, also called rules, are used to verify the consistency of the ontology and to perform inferences.

The use of ontology empowers the execution of some interesting features, such as semantic reasoners and semantic queries. Semantic reasoners, for example Pellet [7], provide the functionalities of *consistency checking*, *concept satisfiability*, *classification* and *realisation*. *Consistency checking* ensures that an ontology does not contain contradictory facts; *concept satisfiability* checks if it is possible for a concept to have instances; *classification* computes the subclass relations between every named class to create the complete class hierarchy; and *realisation* finds the most specific classes that an individual belongs to [7]. In other words, semantic reasoners are able to infer logical consequences from a set of axioms. Reasoners are also used to apply rules such as the ones coded in SWRL (Semantic Web Rule Language). Moreover, ontologies can be semantically queried through SQWRL (Semantic Query-enhanced Web Rule Language), which is a simple and expressive language for implementing semantic queries in OWL [8]. OWL is a semantic web standard formalism intended to explicitly represent the meaning of terms in vocabularies and the relationships between those terms [1].

OWL is based on Description Logics (DL), which formed the basis of several well-known ontology languages [6]. The name DL is motivated by the fact that the important notions of the domain are specified by concept descriptions, *i.e.*, expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. DLs are usually equipped with a terminological and an assertional formalism [6]. On one hand, terminological axioms can be used to introduce names (abbreviations) for complex descriptions, and a set of terminological axioms is called a TBox. On the other hand, the assertional formalism can be used to state properties of individuals. A set of such assertions is called an ABox, and the named individuals that occur in ABox assertions are called ABox individuals [6]. DL systems provide their users with various inference capabilities that deduce implicit knowledge from the explicitly represented knowledge [6]. For example, the *subsumption algorithm* determines subconcept-superconcept relationships; the *instance algorithm* infers instance relationships; and the *consistency algorithm* identifies whether a knowledge base (consisting of a set of assertions and a set of terminological axioms) is non-contradictory. Therefore, the classes, properties and axioms of an ontology compose its TBox (Terminological Box), while the individuals and relationships comprise its ABox (Assertion Box).

OWL is a language based on DL, and as such uses first-order predicate logic as its underlying knowledge representation and reasoning paradigm. OWL is a language for processing web information that became a W3C recommendation in February 2004 [1]. W3C classifies OWL into three sublanguages, each of which is intended to supply different aspects affecting their expressiveness and inference complexity (*i.e.*, performance): OWL Lite, OWL DL and OWL Full [9]. **OWL Lite** is the most simple and restricted version of OWL in terms of expressiveness. **OWL DL** is so called because it uses Description Logic to represent the relations between objects and their properties, providing maximum expressiveness while preserving the completeness of computational properties. Finally, **OWL Full** provides highest expressiveness and the syntactic freedom of RDF (Resource Description Framework) but without preserving guarantees on computational complexity.

OWL basic components are classes, properties and individuals. We can say that a class is disjoint from other classes using the *owl:disjointWith* element, and equivalence between classes can be defined using a *owl:equivalentClass* element. Considering the definition of concepts, suppose we wish to declare that the class *C* satisfies certain conditions, that is, every instance of *C* satisfies these restrictions, and/or that every instance that satisfies these restrictions can be inferred as belonging to *C*. OWL provides the following elements to represent these class conditions [9]: *owl:allValuesFrom* to define the class of possible values that the property can take (in terms of logic, it is an *universal quantification*, *i.e.*, all values of the property must come from this class); *owl:minCardinality* to represent a *cardinality restriction*, requiring a minimum number of relationships (it is the opposite of the *owl:maxCardinality*, which imposes an upper limit of relationships); *owl:someValuesFrom* to represent the *existential quantification*; and *owl:hasValue* to state that the property must have a specific value. OWL was defined with two kinds of properties [9]: *object properties*, which relate objects (instances of classes, that is, interesting elements in the domain of discourse) to other objects; and *datatype properties*, which relate objects to datatype values. Also, OWL allows the definition of some characteristics of property elements directly [9], such as if the property is transitive, symmetric, functional, and so on.

Ontologies and rules are two established paradigms in knowledge modelling [10], and OWL ontologies can be combined with rules, such as the Semantic Web Rule Language (SWRL) [11]. To improve the expressiveness of OWL ontologies, SWRL was proposed as a rule extension of OWL DL that adheres to the open-world paradigm. SWRL adds to the expressive power of OWL by allowing the modelling of certain axioms which lie outside the capability of OWL DL [10]. SWRL includes a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL. The proposed rules are of the form of an implication between an antecedent (body) and a consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold (be true). A rule has the form [11]:

$$antecedent \Rightarrow consequent$$

where both antecedent and consequent are conjunctions of atoms written $a1 \wedge ... \wedge an$. Variables are indicated using the standard convention of prefixing them with a question mark (*e.g.*, ?x). Using this syntax, a rule asserting that the composition of parent and brother properties implies the uncle property would be written as follows [11]:

$$parent(?x, ?y) \wedge brother(?y, ?z) \Rightarrow uncle(?x, ?z)$$

Both the antecedent and the consequent of a rule might consist of zero or more atoms. On one hand, an empty antecedent is treated as trivially true (*i.e.*, satisfied by every interpretation), so the consequent must also be satisfied by every interpretation. On the other hand, an empty consequent is treated as trivially false (*i.e.*, not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction [11]. A SWRL atom can be [10] unary (such as a class expression) or binary (such as an object property). Moreover, the arguments in atoms are of the form of individuals or data values.

This section presented a background on ontologies, where we can observe that several advantages can emerge by using them more expressively in planning, *i.e.*, combining ontologies with planning. Next section investigates the state of the art regarding related researches that currently integrate ontologies with planning.

3 Related Work

The work in [12] demonstrates how an OWL reasoner can be integrated with an artificial intelligence planner. Investigations on the efficiency of such an integrated system and how OWL reasoning can be optimized for this context were also presented. In their approach, the reasoner is used to store the world state, answer the planner's queries regarding the evaluation of preconditions, and update the state when the planner simulates the effects of operators. Also, they described the challenges modelling service preconditions, effects and the world state in OWL, examining the impact of this in the planning process. Specifically, the SHOP2 HTN planning system was integrated with the OWL DL reasoner Pellet to explore the use of semantic reasoning over the ontology [12].

A generic task ontology to formalise the space of planning problems was proposed by [13]. According with its authors, this task ontology formalises the nature of the planning task independently of any planning paradigm, specific domains, or applications and provides a fine-grained, precise and comprehensive characterization of the space of planning problems. The OCML (Operational Conceptual Modelling Language) was used to formalise the task ontology proposed in [13], since it was argued that this language provides both support for producing sophisticated specifications, as well as mechanisms for operationalising definitions to provide a concrete reusable resource to support knowledge acquisition and system development.

Another related work [14] defines a series of translations from ontologies to planning formalisms: one from OWL-S process models to SHOP2 domains; and another from OWL-S composition tasks to SHOP2 planning problems. They implemented and described a system which performs these translations, using an extended SHOP2 implementation to plan with over the translated domain, and then executing the resulting plans. In summary, the work of [14] explored how to use the SHOP2 HTN planning system to do automatic composition in the context of Web Services described in OWL-S ontologies.

The work of [15] proposes a planning and knowledge engineering framework based on OWL ontologies that facilitates the development of domains and the use of Description Logic (DL) reasoning during the planning steps. In their model, the state of the world is represented as a set of OWL facts (*i.e.*, assertions on OWL individuals), represented in an RDF (Resource Description Framework) graph; actions are described as RDF graph transformations; and planning goals are described as RDF graph patterns. Their planner integrates DL reasoning by using a two-phase planning approach that performs DL reasoning in an off-line manner, and builds plans on-line, without doing any reasoning. Their planner uses a subset of DL known as DLP (Description Logic Programs) that has polynomial time complexity and can be evaluated using a set of logic rules.

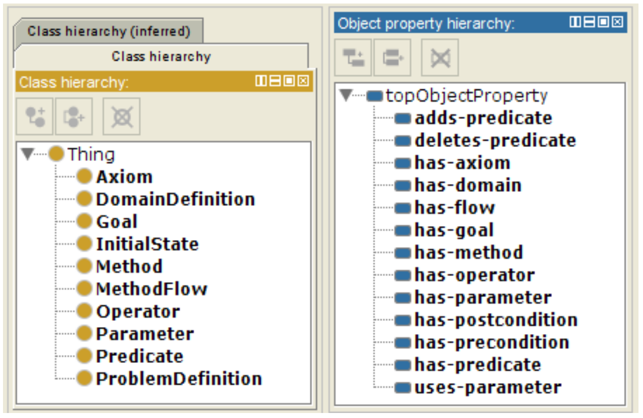
We can observe from these related work that several authors are proposing semantic representation of planning domains in ontologies. Also, approaches to translate from planning formalisms to ontologies and vice-versa are usually explored, so as the use of semantic reasoners before or during the planning steps. However, to the best of our knowledge, our work is the first to address the integration of ontologies in OWL with the HTN planning formalism to propose the ontology that will be presented in next section.

4 The HTN Ontology Conceptualisation

In classical planning, the main aim of the planning task is to attain a goal-state, which is usually specified in terms of a number of desired properties of the world. To model this domain, we developed an ontology, encoded in OWL [1] and built with ProtÃľgÃľ, to represent HTN planning domains. ProtÃľgÃľ is an open source ontology editor which also enables the visualisation of ontologies in different ways, the execution of semantic reasoners, and several other interesting features. The concepts and properties modelled in our proposed HTN planning ontology can be visualised in Figure 1. The conceptualisation was created based on the definitions of [2], [16] and [4], and a description of these concepts can be found next:

- **DomainDefinition:** A domain description is a description of a planning domain, consisting of a set of methods, operators, and axioms.
- **Operator:** Each operator indicates how a primitive task can be performed. It is composed of: name, parameters, preconditions, a delete list and an add list giving the operator's negative and positive effects.
- **Method:** Each method indicates how to decompose a compound task into a partially ordered set of subtasks, each of which can be compound or primitive. The simplest version of a method has three parts: the task for which it is to be used, the preconditions, and the subtasks that need to be done in order to accomplish it.
- **Axiom:** Axioms can infer preconditions that are not explicitly asserted in the current state. The preconditions of methods or operators may use conjunctions, disjunctions, negations, universals and existential quantifiers, implications, numerical computations and external function calls.
- **Predicate:** A predicate has a name and it contains any number of parameters. Predicates are used to represent the preconditions and postconditions of actions, as well as the state of the world (*i.e.*, the state of affairs).
- **Parameter:** A parameter is a variable symbol whose name begins with a question mark (*e.g.*, as ?x or ?agent), and it is used by operators, methods and predicates.
- **MethodFlow:** Each method must contain at least one flow, and each flow has a specific position. A method flow contains an ordered list of preconditions and an ordered list of methods or operators invocations.

- **ProblemDefinition:** Planning problems are composed of logical atoms (*i.e*, initial state) and task lists (high-level actions to perform), which means, a set of goals.
- **Goal:** Goals in HTN are method invocations with specific parameters that the planner will have to decompose in a sequence of operators (*i.e.*, a plan).
- **InitialState:** An initial state is composed of a set of predicates representing the state of the world at the beginning of the simulation.



- Thing
  - Axiom
  - DomainDefinition
  - Goal
  - InitialState
  - Method
  - MethodFlow
  - Operator
  - Parameter
  - Predicate
  - ProblemDefinition

- topObjectProperty
  - adds-predicate
  - deletes-predicate
  - has-axiom
  - has-domain
  - has-flow
  - has-goal
  - has-method
  - has-operator
  - has-parameter
  - has-postcondition
  - has-precondition
  - has-predicate
  - uses-parameter

Fig. 1. Concepts and properties of the planning ontology

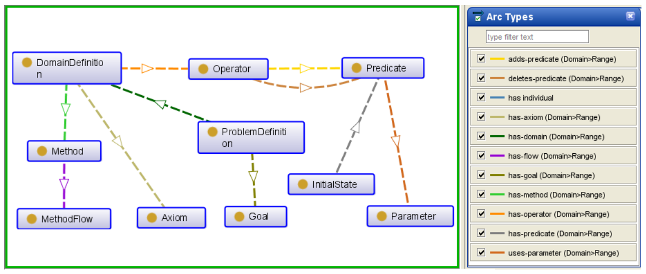
The concepts that are used as domain or range of the properties in the proposed HTN planning ontology are presented in Table 1. Some object properties have only one concept as domain and/or range (*e.g.*, the property *has-operator* has *DomainDefinition* as domain and *Operator* as range). However, it is possible to use logical expressions that include more than one concept in this slot (such as the case of the *has-precondition* property that has the *MethodFlow* concept as domain and the expression "*Operator or Method*" as range). Figure 2 illustrates these properties in a more intuitive way using the OntoGraf plug-in, which can be found in ProtÃl’gÃl’. In this representation, the ontology is viewed as a graph, where the nodes are concepts and the edges represent object properties relating the concepts.

Besides the classes and properties, OWL annotations were used to represent additional information in the relationships of this ontology instantiations. Three new annotations were designed with this purpose, named: *position*, *parameters* and *logicalExpression*. The *position* annotation stores the location where that element must be written in the corresponding jshop file, and it can be used in the following properties: *has-flow, has-precondition, adds-predicate, deletes-predicate, uses-parameter* and *has-parameter*. The *logicalExpression* annotation was created to be used only in relationships involving the *has-precondition* property. Finally, the *parameters* annotation must be used only within the properties *has-precondition, adds-predicate* and *deletes-predicate*. This annotation was employed in order to relate instances of predicates used to define specific operators and methods with instances of parameters.

Table 1. Domain and range of each HTN ontology property

Domain	**Property**	**Range**
DomainDefinition	has-operator	Operator
DomainDefinition	has-method	Method
DomainDefinition	has-axiom	Axiom
InitialState	has-predicate	Predicate
Method	has-flow	MethodFlow
Operator	adds-predicate	Predicate
Operator	deletes-predicate	Predicate
Predicate	uses-parameter	Parameter
ProblemDefinition	has-domain	DomainDefinition
ProblemDefinition	has-goal	Goal
Method, Operator or Predicate	has-parameter	Parameter
MethodFlow	has-precondition	Operator or Method
MethodFlow or Operator	has-postcondition	Predicate

This section presented how we model the concepts and properties of our HTN planning ontology using OWL. Next section shows an instantiation (ABox) of this previously explained ontology conceptualisation (TBox).



☑ adds-predicate (Domain>Range)
☑ deletes-predicate (Domain>Range)
☑ has individual
☑ has-axiom (Domain>Range)
☑ has-domain (Domain>Range)
☑ has-flow (Domain>Range)
☑ has-goal (Domain>Range)
☑ has-method (Domain>Range)
☑ has-operator (Domain>Range)
☑ has-predicate (Domain>Range)
☑ uses-parameter (Domain>Range)

Fig. 2. Visual representation of the planning ontology

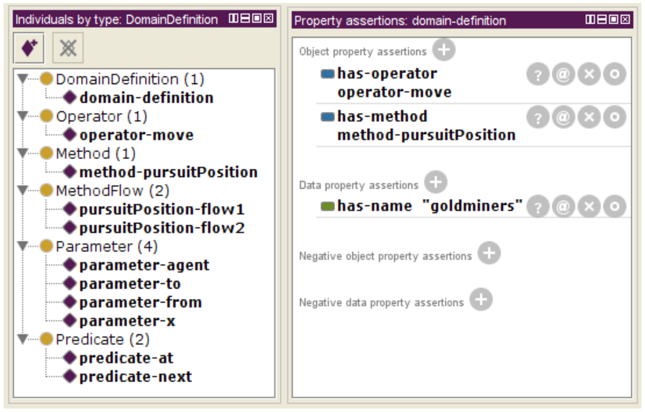
5 Instantiating the HTN Ontology

To investigate the feasibility of defining an HTN planning domain as an instantiation of our OWL ontology, we used the ProtÃl'gÃl' ontology editor to create a simple HTN problem domain scenario. The following code illustrates the corresponding HTN domain definition (named *goldminers*) that includes only one operator (named *move*) and one method (named *pursuitPosition*). The operator *move* has two preconditions, one negative effect and one positive effect, all represented as predicates. The method *pursuitPosition* has two different flows, each one with its corresponding preconditions and effects.

```
( defdomain goldminers (
( :operator (!move ?agent ?from ?to)
((at ?agent ?from) (next ?from ?to))
((at ?agent ?from))
((at ?agent ?to)))

( :method (pursuitPosition ?agent ?from ?to)
((at ?agent ?from) (next ?from ?to))
((!move ?agent ?from ?to))
((at ?agent ?from) (next ?from ?x))
((!move ?agent ?from ?x) (pursuitPosition ?agent ?x ?to)))
)
```

A snapshot of the instantiation using the previously presented scenario (*goldminers*) can be seen in Figure 3. It is important to highlight that Figure 3 illustrates the ontology instantiation in ProtÃl’gÃl’ that corresponds exactly to the previously explained HTN specification. Thus, it is possible to convert from the ontology formalism to the planning specification. In fact, the two implemented methods, one for converting from HTN to OWL and the other to translate from OWL to HTN, will be explained later in this paper.



Individuals by type: DomainDefinition

- DomainDefinition (1)
  - domain-definition
- Operator (1)
  - operator-move
- Method (1)
  - method-pursuitPosition
- MethodFlow (2)
  - pursuitPosition-flow1
  - pursuitPosition-flow2
- Parameter (4)
  - parameter-agent
  - parameter-to
  - parameter-from
  - parameter-x
- Predicate (2)
  - predicate-at
  - predicate-next

Property assertions: domain-definition

Object property assertions
has-operator operator-move
has-method method-pursuitPosition

Data property assertions
has-name "goldminers"

Negative object property assertions

Negative data property assertions

Fig. 3. Instantiating the planning ontology according to the goldminers specific planning domain

One advantage of using an ontology editor is the capability of enhancing the graphic visualisation of problem domains instances and their relationships, as illustrated in Figure 4. This visualisation was obtained using a ProtÃl’gÃl’ plugin known as OntoGraf. However, it is possible to explore the ontologies using different approaches and editors. Moreover, an ontology representation makes possible to explore features such as rules in SWRL and inferences empowered by semantic reasoners. Next section shows how to convert from our planning ontology in OWL to specifications used by artificial intelligence planners such as JSHOP.

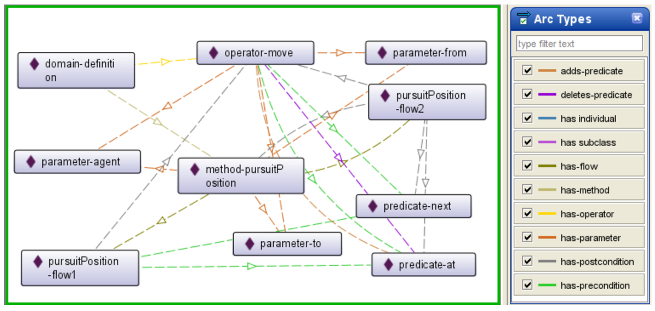


Fig. 4. Visualising the instances of the planning ontology

6 Planning and Ontology Conversions

This section demonstrates, in a high level of abstraction, the algorithms implemented in Java to convert OWL ontologies to HTN specification files, and vice-versa, which is from HTN domain definitions to the corresponding OWL ontology instances. Thus, we established a bidirectional mapping among the elements of our OWL planning ontology and the elements represented in the HTN domain specifications.

6.1 Converting from the OWL Ontology to HTN

The OWL API [3] was used to read the ontology in Java and write the corresponding jshop file. The instances and properties in the ontology are queried and the corresponding HTN component is generated to that specific ontology element. The concepts, properties and annotations previously presented in this paper are queried to construct the corresponding jshop file. The algorithm for converting the OWL to a jshop file is the following:

```
for each instance $df$ of DomainDefinition concept do
    create the jshop corresponding file
    $operators \leftarrow$ has-operator relationships of $df$
    for each Operator $op$ in $operators$ do
        extract $op$ information from the ontology
        write $op$ parameters, conditions and effects in order
    end for
    $methods \leftarrow$ has-method relationships of $df$
    for each Method met in methods do
        extract met information from the ontology
        write met parameters and flows in order
    end for
end for
```

6.2 Converting from HTN to the OWL Ontology

The OWL API [3] was also used to write the ontology, after implementing a parser in Java to read and interpret the jshop file. This approach makes the opposite direction from the previous one, which converted from the OWL planning ontology to an HTN specification. In this approach, for each component found when parsing the jshop file, such as a new operator, method or axiom, then the equivalent OWL individual is created with the OWL API and included in the ontology instantiation being created. It is important to note that some components become instances, but others become object properties, data properties or annotations. The algorithm for converting the jshop file to a corresponding OWL is the following:

```
while there are tokens remaining in the jshop file do
    $token \leftarrow nextToken()$
    if $token$ = defdomain then
        create corresponding DomainDefinition instance
    end if
    if $token$ = operator then
        create corresponding Operator instance
        read its parameters, preconditions and effects
        create the corresponding ontology elements
    end if
    if $token$ = method then
        create corresponding Method instance
        read its parameters and flows
        create the corresponding ontology elements
    end if
end while
```

7 Final Remarks

This paper proposed an HTN (Hierarchical Task Network) ontology in order to represent planning formalisms using semantic technologies. More specifically, we presented an ontology coded in OWL to represent HTN domains and problems in the context of automated planning. The proposed ontology was instantiated using ProtÃľgÃľ to exemplify how it can be used and to demonstrate its feasibility. Also, we presented algorithms to convert specifications from OWL to HTN, and vice-versa, that were coded in

Java using the OWL API [3]. As pointed out in [15], the use of OWL ontologies as a basis for modelling domains allows the reuse of existing knowledge in the semantic web. However, research in this direction is still in their initial steps yet. We briefly shown the state of the art of approaches that integrate ontologies with planning, commenting their contributions.

As future work, it would be interesting to investigate further ontology reasoning mechanisms and semantic technologies features within the scope of the proposed HTN ontology. One example would be creating rules (*e.g.*, in SWRL - Semantic Web Rule Language) to infer new knowledge such as inconsistencies in this ontology instantiation. Thus, as next step in this direction, we are going to explore advantages of using semantic representations of planning domains, such as the reasoning enabled by ontologies. The ability to use ontologies to infer and generate knowledge over a domain is a motivation to continue investigating integrations of ontology representations with planning.

This work demonstrated new possibilities that can be explored in the direction of integrating the areas of ontologies and automated planning. Given the similarities among HTN planning and agent programming plans, we will also explore as future work how to convert from this ontology to automatically generate a corresponding AgentSpeak code, which is a logical language to program agent plans. As examples of relations between concepts in these two formalism we can currently highlight: method & plan; precondition & context; and operator & external action. Thus, we also want to investigate and develop algorithms to convert from this OWL ontology to AgentSpeak plans, and vice-versa.

Another interesting possibility to explore based in our work is extending the planning ontology to address further planning characteristics, such as non deterministic HTN planning formalisms. However, if the conceptualisation changes, the parsers may have to be adjusted accordingly to handle the new concepts and properties in the ontology. Currently, we plan to continue assessing the correctness of our algorithms (for converting from HTN to OWL and from OWL to HTN) by further testing them with more complex examples.

8 Acknowledgements

Will appear in final version.

References

1. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. Technical report, W3C (February 2004)
2. Erol, K., Hendler, J.A., Nau, D.S.: HTN planning: Complexity and expressivity. In Hayes-Roth, B., Korf, R.E., eds.: AAAI, AAAI Press / The MIT Press (1994) 1123–1128
3. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semant. web **2**(1) (January 2011) 11–21
4. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: an HTN planning system. J. Artif. Int. Res. **20**(1) (December 2003) 379–404

5. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum.-Comput. Stud. **43**(5-6) (December 1995) 907–928
6. Baader, F., Horrocks, I., Sattler, U.: Description logics. In: Handbook on Ontologies. (2004) 3–28
7. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. Web Semant. **5**(2) (June 2007) 51–53
8. O'Connor, M.J., Das, A.K.: SQWRL: a query language for OWL. In Hoekstra, R., Patel-Schneider, P.F., eds.: OWLED. Volume 529 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
9. Antoniou, G., van Harmelen, F.: Web Ontology Language: OWL. In: Handbook on Ontologies. (2004) 67–92
10. Hitzler, P., Parsia, B. In: Ontologies and rules. Springer (2009) 111–132
11. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language combining OWL and RuleML. W3c member submission, World Wide Web Consortium (2004)
12. Sirin, E., Parsia, B.: Planning for semantic web services. In: Semantic web services workshop at 3rd international semantic web conference (iswc2004). (2004)
13. Rajpathak, D., Motta, E.: An ontological formalization of the planning task. In: International Conference on Formal Ontology in Information Systems (FOIS 2004). (2004) 305–316
14. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Web Semant. **1**(4) (October 2004) 377–396
15. Bouillet, E., Feblowitz, M., Liu, Z., Ranganathan, A., Riabov, A.: A knowledge engineering and planning framework based on OWL ontologies. In: Proceedings of the Second International Competition on Knowledge Engineering. (2007)
16. Ilghami, O.: Documentation for JSHOP2. Technical report, University of Maryland, Department of Computer Science, College Park, MD 20742, USA (May 2006)