Semantic Representations of Agent Plans and Planning Problem Domains

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

Pontifical Catholic University of Rio Grande do Sul - PUCRS
Postgraduate Programme in Computer Science, School of Informatics (FACIN)
Porto Alegre - RS - Brazil
{artur.freitas,daniela.schmidt,alison.panisson}@acad.pucrs.br,
{felipe.meneguzzi,renata.vieira,rafael.bordini}@pucrs.br

Abstract. Integrating knowledge representation approaches with agent programming and automated planning is still an open research challenge. To explore the combination of those techniques, we present a semantic model of planning domains that can be converted to both agent programming plans as well as planning problem definitions. Our approach allows the representation of agent plans using ontologies, enabling the integration of different formalisms since the knowledge in the ontology can be reused by several systems and applications. Ontologies enable the use of semantic reasoning in planning and agent systems, and such semantic web technologies are significant current research trends. This paper presents our planning ontology, exemplify its use with an instantiation, and shows how to translate between ontology, agent code, and planning specifications. Algorithms to convert between these formalisms are shown, and we also discuss future directions towards the integration of semantic representation, automated planning, and agent programming.

Keywords: ontology, knowledge representation, agent plan, automated planning

1 Introduction

Knowledge representation approaches using ontologies are being studied as promising techniques to enable semantic reasoning, knowledge reuse, interoperability, and so on. However, the use of ontologies integrated with agent systems and planning formalisms is still a research path at its initial steps. To investigate this issue, we present a semantic model to represent the knowledge about planning domains.

More specifically, we developed an ontology encoded in OWL (Web Ontology Language) [1] to model planning domains based on the HTN (Hierarchical Task Network) paradigm [2]. This conceptualisation was instantiated in the Protégé[1] ontology editor to model a classical problem, known as "Gold Miners". This example demonstrates how planning domains can be modelled in our ontology, and we also show the equivalent agent plans and planning specifications generated from this scenario.

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

Furthermore, we propose algorithms to convert the OWL planning ontology to different formalisms, such as agent programming plans in AgentSpeak [3] and planning problem domain specifications in SHOP (Simple Hierarchical Ordered Planner) [4]. These algorithms to automatically translate from OWL to other formalisms (and vice-versa) were implemented in Java using the OWL API [5]. Therefore, planning domains instantiated in the ontology can be automatically converted to AgentSpeak [3] or SHOP [4] code (and the other way around) using the 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 beginning to emerge in the relevant communities.

For instance, our approach enables to derive planning domain models and agent programming plans from existing ontological knowledge, and also to convert again from these formalisms to ontology representations. In other words, this work investigates the integration of ontologies with agent programming and other planning formalisms in order to explore semantic representations of planning domains. Thus, our goal is to explore and demonstrate the utilisation of ontologies more expressively than previous work in automated planning and agent-oriented development.

This paper is organised as follows. Next section provides a comprehensive background on ontologies, focusing on preparing the reader to relate ontologies with agent-oriented programming and planning formalisms. A section of related work is presented afterwards to map the state of the art on using ontologies in planning systems. Then, a section explaining our conceptualisation (TBox, *i.e.*, Terminological Box) is presented. This conceptualisation is composed of classes and properties to represent planning domains. 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 planning problem. We explain how to convert from our planning ontology to AgentSpeak [3] plans; and also from the ontology to SHOP [4] domain definitions. Algorithms coded in Java with the OWL API [5] to make these conversions are discussed afterwards. Then, we conclude this paper and point out other possible investigations and research directions towards the integration of ontology, planning and agent development.

2 Ontologies and OWL

Ontology is defined as an "explicit specification of a conceptualisation" [6]. A conceptualisation stands for an abstract model of some aspect of the world, therefore an ontology is a knowledge representation structure composed of concepts, properties, individuals, relationships and axioms [7], as described in sequence. A **concept** is an abstract group, set, class or collection of objects that share common properties. 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. An **individual** (also called instance, object or fact) is the "ground-level" component of an ontology which represents a specific element of a concept or class. A **relationship** is an instance of a property, which relates two individuals: one in the relationship domain, and one in its range. 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 [8], 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 [8]. 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) [9]. Moreover, ontologies can be semantically queried through SQWRL (Semantic Query-enhanced Web Rule Language) [10], which is a simple and expressive language for implementing semantic queries in OWL. 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 ontology languages [7]. 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. DL systems provide various inference capabilities to deduce implicit knowledge from the explicitly represented knowledge [7]. 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.

Given this technological development, it is natural to think that there would be many advantages in using it more expressively in agent-oriented software engineering. The work reported in [11] pointed out to the following advantages of such integration: (*i*) more expressive queries in the belief base, since its results can be inferred from the ontology and thus are not limited to explicit knowledge; (*ii*) refined belief update given that ontological consistency of a belief addition can be checked; (*iii*) the search for a plan to deal with an event is more flexible because it is not limited to unification, *i.e.*, it is also possible to consider subsumption relationships between concepts; and (*iv*) agents can share knowledge using ontology languages, such as the case of OWL.

This section presented a background on ontologies, where we can observe that several advantages can emerge by using them more expressively in agent-oriented software engineering and planning. Next section investigates the state of the art regarding related studies integrating ontologies with artificial intelligence planning approaches.

3 Related Work

The work in [12] explains how an OWL reasoner can be integrated with an artificial intelligence planner. Investigations on the efficiency of such 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 of 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 in [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 describe an implemented 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.

Reference [15] proposes a planning and knowledge engineering framework based on OWL ontologies that facilitates the development of domains and uses 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.

Several authors are proposing semantic representation of planning domains in ontologies. Also, approaches to translate among planning formalisms and ontologies are usually explored. These approaches can involve 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 [1] with both the HTN [2] formalism and with agent programming plans.

Next section explains the proposed planning ontology coded in OWL [1], which is explored to generate both agent plans in AgentSpeak [3] and SHOP [4] specifications of planning problem domains.

4 The Planning 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 Protégé, to represent HTN planning domains. Protégé 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 formalized in our proposed HTN planning ontology can be visualised in Figure 1. The conceptualisation was created based on the definitions of [2], [16] and [17], and a description of these concepts can be found next:

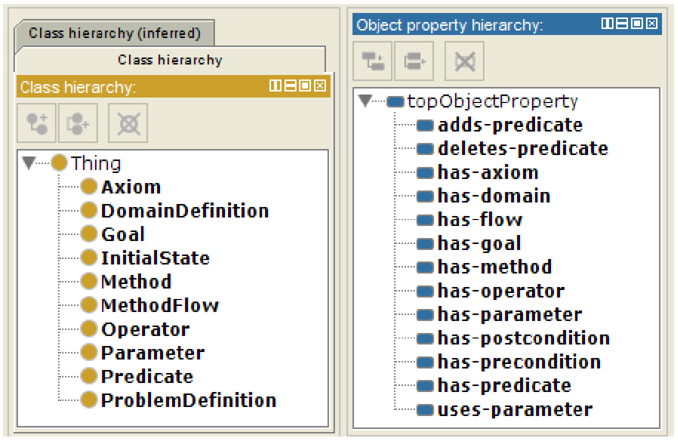


Fig. 1. Concepts and properties of the planning ontology

– **DomainDefinition:** A domain definition 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:** The flows of a method specify how it can be decomposed based on the current state of the world (which is represented in predicates). Thus, each method flow contains an ordered list of preconditions and an ordered list of methods or operators invocations. Each method must contain at least one flow.
– **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 instance of initial state models the problem by means of predicates that represent the state of the world at the beginning of the simulation.

The concepts that are used as domain or range of each property in the proposed HTN planning ontology are presented in Table 1. This table illustrates formal definitions that were developed to formalize the knowledge represented in our ontology. 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, logical expressions were also used to include more than one concept in this slot, such as the case of the *has-postcondition* property that has the *MethodFlow* concept as domain and the expression "*Operator or Method*" as range.

Table 1. Domain and range of each property in the planning ontology

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 or Operator	has-precondition	Predicate
MethodFlow	has-postcondition	Operator or Method

Besides the classes and properties, OWL annotations were used to represent additional information in the relationships of this ontology instantiations. When representing relationships with predicates or parameters, the order in which they have to appear must be known, which is annotated when a property targeting one of them is instantiated. Annotations are also the best choice to model logical expressions among predicates and which parameters are required when a method or operator instance relates with a predicate. Three new annotations were designed with this purpose: *position*, *logicalExpression* and *parameters*. The *position* annotation stores the location where that element must be written in the corresponding files, 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.

Figure 2 illustrates the concepts and properties (with their domain and range) in a more intuitive way using the OntoGraf[2] plug-in, which can be found in Protégé. In this representation, the ontology is viewed as a graph, where the nodes are concepts and the edges represent object properties relating the concepts. This section presented how we modelled the concepts and properties of our HTN planning ontology using OWL. The next sections show an instantiation (ABox) of this previously explained ontology conceptualisation (TBox) to model a specific scenario. Then, we show the equivalent agent programming plans in AgentSpeak [3] and planning domain specifications in SHOP [4] derived from our ontology representation.

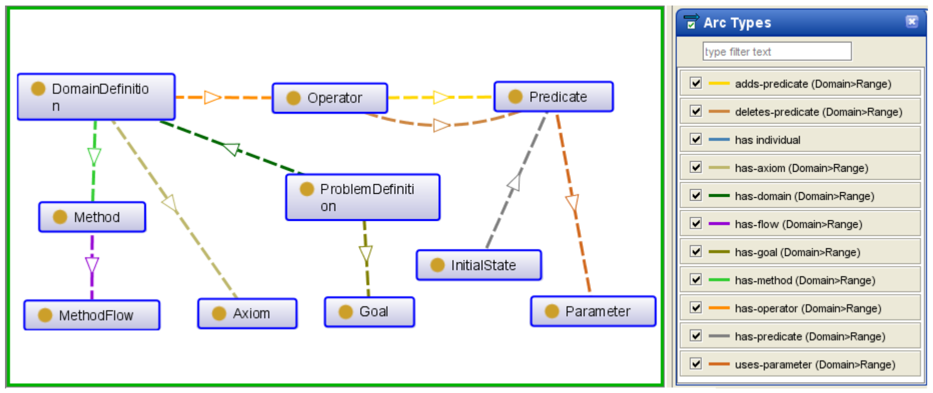


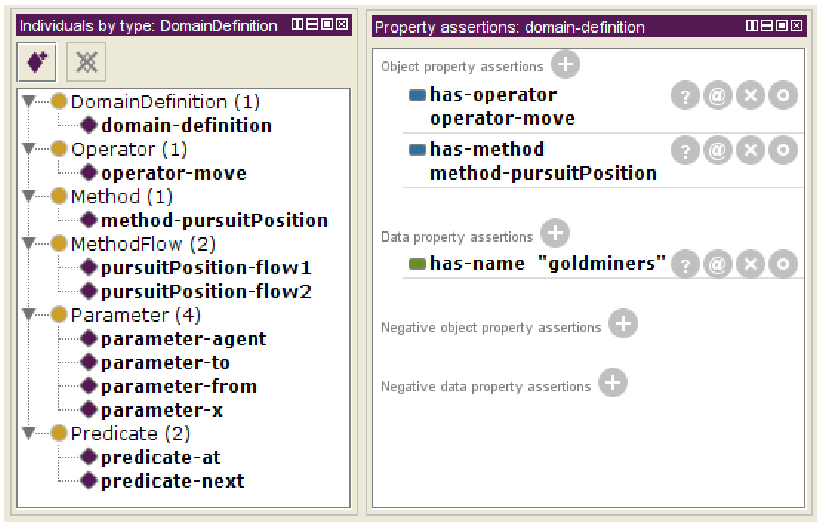
Fig. 2. Visual representation of our planning ontology in Protégé (OntoGraf plug-in)

5 Instantiating the Planning Ontology

To investigate the feasibility of defining a planning domain as an instantiation of our OWL ontology, we also used the Protégé ontology editor to create a simple definition of

[2] http://protegewiki.stanford.edu/wiki/OntoGraf

a planning problem domain scenario. We modelled a well-known multi-agent scenario known as *gold miners*[3], where agents playing the role of miners have to move in an environment, and search specific positions. Our scenario includes only one instance of the *Operator* concept (named *move*) and one instance of *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. A snapshot of the instantiation using this scenario (*gold miners*) can be seen in Figure 3. It is important to highlight that Figure 3 illustrates the ontology instantiation in Protégé that corresponds exactly to the previously explained specification. Next we demonstrate that it is possible to convert from our ontology formalism both to planning specifications and to agent plans. In fact, this paper explains methods for converting among these different formalisms.


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 our planning ontology according to the goldminers specific planning domain

An advantage of using ontology editors is the capability of enhancing the graphic visualisation of planning problem domains instances as well as agent plans and their relationships, as illustrates Figure 4. This visualisation was obtained using a Protégé plug-in known as OntoGraf, however it is possible to explore the ontologies using different editors. In this example, the user can visualize domain features such as how the instances are related, and the visualization can be customized to show only the desired characteristics of the corresponding instantiation. Moreover, an ontology representation makes possible to explore features such as rules coded in SWRL [9] and inferences empowered by semantic reasoners [8]. The next sections show how to convert from our

[3] http://multiagentcontest.org/2006

planning ontology in OWL both to agent programming plans in AgentSpeak [3] and to artificial intelligence planners specifications in SHOP [4].

The list of instances and their relationships is presented below, where “a : C” denotes that the instance ‘a’ is a type of ‘C’, and “(a,b) : R” indicates that the instance ‘a’ is related to instance ‘b’ through the property ‘R’. This list is a full description of the example used in this paper, which corresponds to Figures 3 and 4. This example was instantiated in the ontology to be converted both to a planning specification in SHOP and agent plans in AgentSpeak.

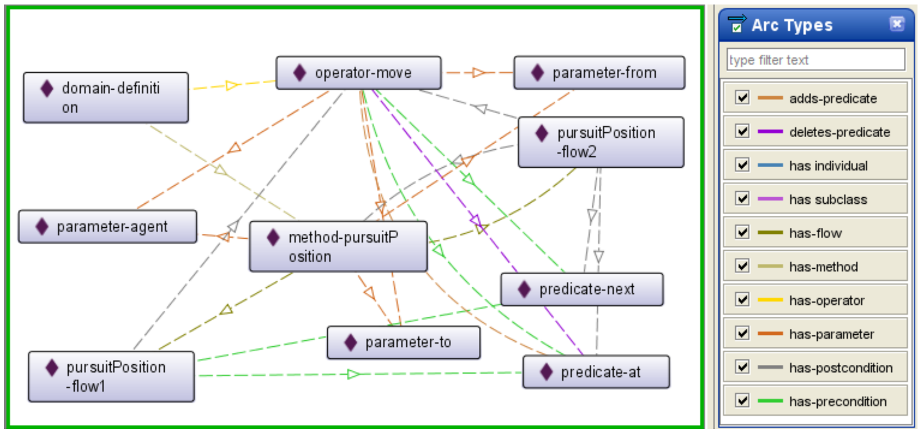


Fig. 4. Visualising the instances of our planning ontology in Protégé (OntoGraf plug-in)

domain-definition : DomainDefinition
operator-move : Operator
method-pursuitPosition : Method
pursuitPosition-flow1 : MethodFlow
pursuitPosition-flow2 : MethodFlow
parameter-agent : Parameter
parameter-to : Parameter
parameter-from : Parameter
parameter-x : Parameter
predicate-at : Predicate
predicate-next : Predicate
(domain-definition, operator-move) : has-operator
(domain-definition, method-pursuitPosition) : has-method
(operator-move, parameter-agent) : has-parameter
(operator-move, parameter-from) : has-parameter
(operator-move, parameter-to) : has-parameter
(operator-move, predicate-at) : has-precondition
(operator-move, predicate-next) : has-precondition
(operator-move, predicate-at) : deletes-predicate

(operator-move, predicate-at) : adds-predicate
(method-pursuitPosition, pursuitPosition-flow1) : has-flow
(method-pursuitPosition, pursuitPosition-flow2) : has-flow
(method-pursuitPosition, parameter-agent) : has-parameter
(method-pursuitPosition, parameter-from) : has-parameter
(method-pursuitPosition, parameter-to) : has-parameter
(pursuitPosition-flow1, predicate-at) : has-precondition
(pursuitPosition-flow1, predicate-next) : has-precondition
(pursuitPosition-flow1, operator-move) : has-postcondition
(pursuitPosition-flow2, predicate-at) : has-precondition
(pursuitPosition-flow2, predicate-next) : has-precondition
(pursuitPosition-flow2, operator-move) : has-postcondition
(pursuitPosition-flow2, method-pursuitPosition) : has-postcondition

Besides the relationships listed above to describe the example instantiated in our ontology, there is a data property *has-name*. Also, our instantiation represent *positions* and *parameters* as annotation in these relationships.

5.1 Converting from our OWL Planning Ontology to AgentSpeak Plans

Most techniques for Multi-Agent System development are heavily inspired by the BDI architecture (Beliefs, Desires and Intentions). For example, the AgentSpeak [18] language was introduced in 1996 as a formalisation of BDI agents to enable agent programs to be written using a notation similar to (guarded) Horn clauses. Agents achieve their goals through the use of plans that can be composed of sub-plans and that are ultimately converted into actions. This approach is similar to the one used in the HTN planning formalism, where methods are decomposed into operators. A plan body coded in AgentSpeak [3] is typically a sequence of actions to be executed and further goals to be achieved. AgentSpeak plans have three distinct parts [3]: the *triggering event*, the *context*, and the *body*. Together, the *triggering event* and the *context* are called the head of the plan. The three plan parts are syntactically separated by ‘:’ and ‘<–’ as follows:

Syntax of AgentSpeak Plans

```
triggering_event : context <- body.
```

The following code (*miner.asl*) corresponds to a plan in AgentSpeak generated from our planning ontology instantiation. The scenario is the *gold miners* previously explained, and this example respects the presented AgentSpeak plan syntax [3]. Every instance of the *Operator* concept is mapped to an agent plan: its name becomes the *triggering event*, its preconditions form the *context* and its effects becomes the *body*. Similarly, each instance of *Method* is also translated to an AgentSpeak plan, with its corresponding preconditions and decomposition scheme. Both the operators and methods mantain their parameters when being converted from the ontology to agent code.

Our *gold miners* scenario instantiated in the ontology generates the *miner.asl* code which is depicted below. It can be noted that the *move Operator* becomes a plan with the *triggering event +!move(Agent, From, To)*. The *context* of this plan is composed of a conjunction of two instances of *Predicate*: *at(Agent, From)* and *next(From, To)*. The body (or effect) of this plan is to execute the external action *move(Agent, From, To)* in the environment, to remove the belief *at(Agent, From)*, and to add the belief *at(Agent, To)*. Similarly, our scenario depicts how a *Method* in our ontology is converted to an AgentSpeak plan. The main difference from the *Operator* previously explained is that the plan *body* is composed of goals to be achieved by the agent.

miner.asl (AgentSpeak code generated from our planning ontology)

```
+!move(Agent, From, To) :
    at(Agent, From) & next(From, To) <-
        move(Agent, From, To);
        -at(Agent, From);
        +at(Agent, To).

+!pursuitPosition(Agent, From, To) :
    at(Agent, From) & next(From, To) <-
        !move(Agent, From, To).

+!pursuitPosition(Agent, From, To) :
    at(Agent, From) & next(From, X) <-
        !move(Agent, From, X);
        !pursuitPosition(Agent, X, To).
```

The contribution of this section is to sketch how an HTN domain in our ontology can be mapped into an AgentSpeak program (however, detailed translation algorithms and implementation are future work).

5.2 Converting from our OWL Planning Ontology to SHOP Domain Definitions

SHOP is a HTN planning system based on *ordered task decomposition* whose syntax and semantics are given in [4]. In other words, SHOP is a HTN-planner implementation which enables domain-independent automated planning. In HTN planning, the objective is to create a plan to perform a set of tasks (abstract representations of things that need to be done), starting with an initial state-of-the-world. HTN planning is done by problem reduction: planners recursively decompose tasks into subtasks until they reach primitive tasks that can be performed directly by planning operators. A set of methods is required in order to tell the planner how to decompose nonprimitive tasks into subtasks, where each method is a schema for decomposing a particular kind of task into a set of subtasks (provided that the preconditions are satisfied).

We briefly highlight SHOP syntax in the code below to facilitate the understanding of how an instantiation can be converted from our ontology to SHOP specifications. Similarly to our ontology, the SHOP formalism is composed of operators and methods, which can contain preconditions and effects.

Syntax of SHOP Planning Domain Definitions

```
(defdomain domain_name (
   (  :operator (!operator_name ?parameters)
   ((preconditions ?parameters))
   ((negative_effects ?parameters))
   ((positive_effects ?parameters)))

   (  :method (method_name ?parameters)
   ((preconditions ?parameters))
   ((method_or_operator ?parameters)))
   )
```

The following code illustrates the corresponding SHOP domain definition (named *gold miners*) which corresponds to the previous explained scenario instantantied in our ontology as example. We can observe that the instances of *Operator* and *Method* (and its corresponding relationships) are converted in the generated *miner.jshop* specification depicted below. More details about the algorithms to convert from our planning ontology to the SHOP planning domain specifications (and vice-versa) can be found in the next section of this paper.

miner.jshop (SHOP code generated from our planning ontology)

```
(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)))

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

6 Planning and Ontology Conversions

This section demonstrates, in a high level of abstraction, the algorithms implemented in Java to convert OWL ontologies to SHOP specification files, and vice-versa, which is from SHOP 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 SHOP domain specifications. The same principle might be applied to convert among our ontology and AgentSpeak code, such as previously demonstrated with an example in this paper, however algorithms for doing that are not presented in this work.

6.1 Converting from the OWL Ontology to SHOP

The OWL API [5] was used to read the ontology elements and parse each one of them, and Java was used to write them in a corresponding jshop file. OWL API is an open source Java API (Application Programming Interface) for creating, manipulating and serialising OWL ontologies.

The instances, concepts, properties and annotations in the ontology previously presented are queried and the corresponding SHOP component is generated to that specific ontology element to construct the corresponding jshop file. For example, *Operator*'s instances might be related with *Parameter*'s instances through the *has-parameter* property, and with instances of *Predicate* by means of the properties *has-precondition*, *adds-predicate* and *deletes-predicate*. 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 ← 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 ← 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 SHOP to the OWL Ontology

Previous section demonstrated how one example is converted from our ontology both to SHOP specifications and AgentSpeak code. This section shows the algorithms to convert both from the ontology to SHOP domain, and vice-versa, which are already implemented. However, the algorithms to convert between ontology and agent plans are currently being developed, but we already exemplified how this conversion can be made in this paper.

The OWL API [5] was also used to write the ontology elements, 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 a specification in SHOP.

In this algorithm, for each component found when parsing the jshop file, such as a new operator or method, then the equivalent OWL individual is created with the OWL API and included in the ontology instantiation being created (which can be instances, object properties, data properties or annotations). For example, when reading an *Operator*, it is required to extract its parameters, preconditions and effects; however while reading a *Method*, the information to be extracted concerns about its parameters and flows. The algorithm to convert a jshop file to a corresponding instantation of our OWL planning ontology 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

We presented an investigation towards the integration of agent-oriented programming and automated planning with semantic technologies. More specifically, this paper proposed an ontology to represent planning formalisms. Our ontology was developed in OWL [1] to represent HTN [2] domains and problems in the context of automated planning and agent-oriented programming. The proposed ontology was instantiated to exemplify its use and to demonstrate its feasibility. Also, we presented algorithms to convert specifications between different formalisms such as OWL [1] and SHOP [4]. The algorithms have been coded in Java using the OWL API [5].

Given the similarities among planning formalisms and agent programming plans, we also explored how to generate a corresponding AgentSpeak [3] code, which is a logical language to program agent plans. As examples of relations between concepts in these two formalisms we can currently highlight: method & plan; precondition & context; and operator & external action. Thus, we also explored how to convert from our OWL [1] planning ontology to AgentSpeak [3] plans, and vice-versa. In other words, our approach enables new ways to derive both planning specifications and agent code.

As pointed out in [15], the use of OWL ontologies as a basis for modelling domains allows the reuse of knowledge in the semantic web. However, research in this direction is still in their initial steps. We have briefly discussed the state of the art of approaches that integrate ontologies with planning and agent-oriented programming, commenting on their findings and contributions.

As future work, we plan to investigate ontology reasoning mechanisms and semantic technologies features within the scope of our planning ontology. One example would be creating rules (*e.g.*, in SWRL [9]) to infer knowledge such as inconsistencies in ontology instantiations. The ability to use ontologies to infer and generate knowledge over a domain is a motivation to investigate how ontology representations can be integrated with planning and agent-oriented programming. Thus, as next step in this direction, we will explore advantages of using the semantic reasoning enabled by ontologies.

Another interesting area to explore 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 new concepts and properties in the ontology. Currently, we plan to continue assessing the correctness of our algorithms (for converting between OWL [1] to SHOP [4]) by testing them with more examples. Moreover, we are currently coding the algorithms to convert beween the ontology and AgentSpeak [3].

This work investigated the conversion from OWL [1] ontologies to both SHOP [4] and AgentSpeak [3], since these languages are used in our research project, but in a similar way different planning systems and agent programming languages could also be explored. The inclusion of ontology-based semantic technologies in such complex multi-agent platforms is expected to bring together the power of knowledge-rich approaches and complex distributed systems.

Acknowledgements

Part of the results presented in this paper were obtained through research on a project titled "Semantic and Multi-Agent Technologies for Group Interaction", sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian federal law No. 8.248/91.

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. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. John Wiley & Sons (2007)
4. Nau, D., Cao, Y., Lotem, A., Avila, H.M.: SHOP: simple hierarchical ordered planner. In: Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1999) 968–973
5. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semant. web **2**(1) (January 2011) 11–21
6. 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
7. Baader, F., Horrocks, I., Sattler, U.: Description logics. In Staab, S., Studer, R., eds.: Handbook on Ontologies. Springer (2009) 3–28
8. 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
9. 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)
10. 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)
11. Moreira, A.F., Vieira, R., Bordini, R.H., Hübner, J.F.: Agent-oriented programming with underlying ontological reasoning. In: Proceedings of the 3rd international workshop on Declarative Agent Languages and Technologies. DALT'05, Berlin, Heidelberg, Springer-Verlag (2006) 155–170
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)
17. 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
18. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In van Hoe, R., ed.: Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World. MAAMAW '96, Eindhoven, The Netherlands (1996) 42–55