A Knowledge Engineering and Planning Framework based on OWL Ontologies

Eric Bouillet, Mark Feblowitz, Zhen Liu, Anand Ranganathan, Anton Riabov
IBM T.J. Watson Research Center, Hawthorne, NY, USA
ericbou, mfeb, zhenl, arangana, riabov@us.ibm.com

Abstract

In this paper, we describe a domain-independent, general purpose knowledge engineering and planning framework that supports the construction of planning domains and problems based on OWL ontologies, and the integration of the planning process with description logic (DL) reasoning. The use of OWL ontologies as a basis for modeling domains allows the reuse of existing knowledge in the Semantic Web. In this model, the state of the world is represented as a set of OWL facts, represented as an RDF graph. Actions are described as RDF graph transformations. Planning goals are described as RDF graph patterns. Our framework allows a number of developers to create and extend the OWL ontologies and the planning domain in a collaborative manner. We have used our framework for automatically constructing workflows for deployment in stream processing systems.

Introduction

The Semantic Web envisions a world where loosely coupled, independently evolving ontologies provide a common understanding of terms between heterogeneous agents, systems, and organizations. In the past few years, different ontologies have been developed in various domains to capture relevant knowledge, e.g. the GALEN medical ontology (Rector & Horrocks 1997), the NCI cancer ontology(nci), etc. Increasingly, OWL (McGuinness & van Harmelen 2004) has become one of the most popular languages for representing ontologies in the Semantic Web. In this paper, we describe a planning and knowledge engineering framework that supports the construction of planning domains and problems based on OWL ontologies, and the integration of the planning process with description logic (DL) reasoning. The use of OWL ontologies as a basis for modeling domains allows the reuse of existing knowledge in the Semantic Web, while the use of DL reasoning increases the expressiveness of the domain specification. Besides, it allows the use of various tools for editing and validating OWL ontologies.

In our planning model, the state of the world is described by a set of OWL facts (i.e. assertions on OWL individuals). These OWL facts are represented as an RDF graph (Beckett 2004). Actions are described as RDF graph transformations. The precondition of an action is specified as an RDF graph pattern, which can be regarded as a conjunctive query in description logics. The effect of an action is also specified by an RDF graph pattern. Description Logic (DL) reasoning is used to determine if an action can be applied to a given state. An action is viewed as a graph transformation (Baresi & Heckel 2002), that describes how the action transforms the initial state to the final state. Planning goals are also specified as RDF graph patterns. A goal is said to be achieved if the state satisfies the goal. One application of this model described in the paper is for planning in stream processing systems.

We have developed a planner that can construct plans based on domains expressed in OWL, given a goal description. The planner is based on the SPPL model proposed in (Riabov & Liu 2006). The SPPL (Stream Processing Planning Language) model, derived from PDDL, has been shown to exhibit significant improvements in planner scalability for workflow composition in stream processing planning domains. However, it is also capable of planning in STRIPS domains. In addition, it can generate optimal plans according to an additive quality metric, where each action is associated with a quality vector and a cost of execution.

Our planner integrates DL reasoning with planning by using a two-phase planning approach where it performs DL reasoning in an offline manner, and builds plans online, without doing any reasoning. Specifically, the planner uses a subset of DL called DLP (Description Logic Programs (Grosof *et al.* 2003)), which has polynomial time complexity and can be evaluated using a set of logic rules.

Our framework provides a number of mechanisms to enable people of different roles, including domain experts and action specifiers, to create and extend the OWL ontologies and the planning domain in a collaborative manner. It provides mechanisms to validate the domain, and check the composability of different actions (i.e. check if two actions can be executed in a sequence in a plan). End-users can also describe planning goals with the help of terms defined in the ontologies. The use of OWL allows representing the domain in a modular and reusable manner, which allows different people to author different parts of the domain.

In the rest of the paper, we describe our planning and knowledge engineering framework. We introduce the OWL-based planning model and also describe how we model domains for stream processing planning tasks. Then, we briefly describe the planner and the domain construction framework. Finally, we end the paper with some thoughts on the scope of the framework.

Planning Model

A key advantage of our framework is the ability to reuse the knowledge captured in different OWL ontologies in the Semantic Web when describing planning domains and problems. In this section, we formally describe how the state, planning actions and goals can be represented using OWL/RDF.

Preliminary definitions from RDF and OWL.

Ontologies provide a formal description of different terms and how they are related. OWL ontologies describe *concepts* (or *classes*), *properties* and *individuals* (or *instances*). OWL is based on description logics. Description Logic (DL) is typically used to represent and reason about the terminological knowledge of an application domain. In DLs, there are two kinds of facts: “TBox” (terminological) and “ABox” (assertional). In general, the TBox contains sentences describing concepts and properties. For instance, it describes concept hierarchies and the domains and ranges of properties. ABox axioms describe “ground” sentences about individuals (or instances). The ABox describes the concepts to which an individual belongs and its relationship to other individuals.

The set of *RDF Terms* (represented as RDF_T) includes the set of URIs (U) and RDF literals ((RDF_L). An *RDF Triple* is a member of the set $RDF_T \times U \times RDF_T$. An *RDF graph* is a set of RDF triples. The nodes in the graph are subjects and objects, and the edges are labeled by properties. We don’t include blank nodes in this work.

Our system currently supports a tractable subset of OWL DL called Description Logic Programs (DLP)(Grosof *et al.* 2003). DLP lies in the intersection of Description Logics and Horn Logic Programs. In particular, it allows reasoning on the ABox to be done using a set of logic rules.

Representing the state of the world

In our model, the state of the world is described using an RDF graph containing triples that represent OWL ABox assertions. These facts are defined according to a TBox defined in one or more ontologies. Different ontologies can be reused to provide descriptions of classes, properties and individuals in the area of interest.

Model of an action

An action can cause the state of the world to change. The preconditions and effects are modeled as RDF graph patterns. An action is viewed as performing an RDF graph transformation.

For example, consider the drive action in Figure 1, which is the action of driving a truck between two locations in the same city. The PDDL description of the action is shown below:

```
(:action drive
  :parameters (?t ?l1 ?l2 ?c)
  :precondition (and (truck ?t) (location ?l1)
                     (location ?l2) (city ?c)
      (at ?t ?l1) (loc ?l1 ?c) (loc ?l2 ?c))
  :effect (and (at ?t ?l2) (not (at ?t ?l1))))
```

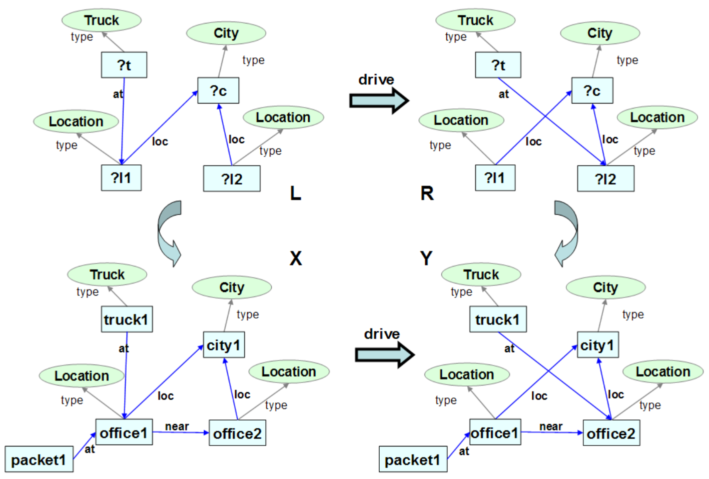

Figure 1: Description of drive action from STRIPS Logistics domain. L and R describe the precondition and effect of the action. X and Y show the state of the world before and after applying the action. Ovals represent OWL classes and rectangles are OWL individuals.

The graph pattern L in Figure 1 describes the precondition of the action and R describes the effect. Note that unary predicates in the above STRIPS representation are modeled as OWL classes, which are connected to variables through the type property (short for the `rdf:type` property in OWL), and binary predicates are represented as OWL properties. n-ary predicates can similarly be converted into RDF by using various strategies such as those described in (Noy & Rector 2004). The RDF graphs X and Y describe possible states of the world before and after applying the action.

We now describe the action model formally. A ***variable*** is a member of the set V where V is infinite and disjoint from RDF_T. A variable is represented with a preceding “?” .

A ***triple pattern*** is a member of the set $(RDF_T \cup V) \times U \times (RDF_T \cup V)$. A ***graph pattern*** is a set of triple patterns.

An ***action*** is of the form $\mathcal{A}(P, E, C, Q)$ where

- P is an RDF graph pattern describing the precondition of the action
- E is an RDF graph pattern describing the effect of the action
- The set of variables in P is a subset of the set of variables in E. This ensures that no free variables exist in the output description.
- C is a cost vector for the action.
- Q is a quality vector for the action.

Conditions for applying an action

An action may be applied if the preconditions are satisfied by the current state. Consider an action $\mathcal{A}(P, E, C, Q)$. Let the current state be G. We define that $\mathcal{P}$ can be applied to G, based on an ontology, O, if and only if there exists a variable substitution function, $(\theta : V \rightarrow RDF_T)$, defined on all the variables in P, such that:

$G \cup O \models \theta(P)$

where $\theta(P)$ is the graph obtained by substituting variables in P based on θ, and $\models$ is an entailment relation defined between RDF graphs, based on description logics. In essence, the graph pattern P is viewed as a conjunctive DL query on the current state of the world.

Effects of applying an action

An action is described as a *graph transformation* operation. Graph transformations have been used in software engineering to describe the behavior of components in terms of transformations of graphs like UML object diagrams (Baresi & Heckel 2002). We adapt these ideas to describe the behavior of components based on transformation of RDF graph patterns.

Let L and R be the action precondition and effect. Now assume that in a plan, L is satisfied by the current state, described by the RDF graph, X. Let θ be the variable substitution function for the variables in L. Let the next state after applying the action be described by the RDF graph, Y. We determine each Y using a graph homomorphism, f, described as:

$f : \theta(L) \cup \theta(R) \rightarrow X \cup Y$

f satisfies the following properties:

1. $f(\theta(L)) \subseteq X$. This follows from the entailment relation between the precondition graph pattern and the state.
2. $f(\theta(R) \subseteq Y$. This means that the next state is a supergraph of the effect graph pattern.
3. $f(\theta(L) \backslash \theta(R)) = X \backslash Y$ and $f(\theta(R) \backslash \theta(L)) = Y \backslash X$ where $\backslash$ represents the graph difference operation. This means that exactly that part of X is deleted which is matched by elements of $\theta(L)$ not in $\theta(R)$, and exactly that part of Y is created that is matched by elements new in $\theta(R)$.

Using properties 2 and 3, it is possible to determine the next state, Y, as a result of applying the action to X. This operation is performed in two main steps. In the first step, we remove all the edges and vertices from X that are matched by $\theta(L) \backslash \theta(R)$ to get a graph D, where $D = X \backslash (\theta(L) \backslash \theta(R))$. We make sure that D is a legal graph, i.e. no edges are left dangling because of the deletion of source or target vertices. In the second step, we glue D with $R \backslash L$ to get Y. This specific mechanism is often referred to as a single pushout graph transformation. Further details about graph transformations can be found in (Rozenberg 1997).

Describing a planning problem

A planning goal is described as an RDF graph pattern. The goal is satisfied by a plan which causes a final state that satisfies the goal RDF graph pattern.

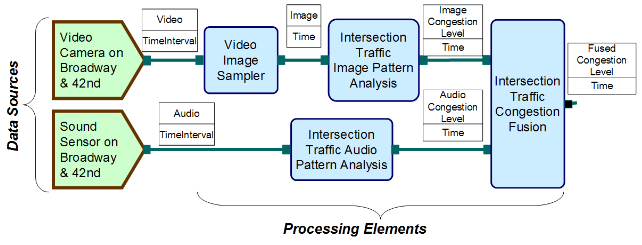
Data Sources
Video Camera on Broadway & 42nd
Sound Sensor on Broadway & 42nd
Video
TimeInterval
Audio
TimeInterval
Video Image Sampler
Image
Time
Intersection Traffic Image Pattern Analysis
Image Congestion Level
Time
Intersection Traffic Audio Pattern Analysis
Audio Congestion Level
Time
Intersection Traffic Congestion Fusion
Fused Congestion Level
Time
Processing Elements

Figure 2: Example Workflow to get real-time traffic congestion levels at an intersection

Component Model for Workflow Composition

The graph transformation model of actions is quite general and can be applied to different kinds of domains. One domain of particular interest to us is automatic construction of workflows from reusable, modular components. The domain we shall describe in this paper is based on stream processing.

Stream processing workflows consist of data sources that produce raw data and processing elements (PEs) that perform operations on the data to produce useful results. The data may be structured, semistructured or unstructured, and may be encoded in different formats. The PEs can perform various kinds of operations like filtering, transformation, classification and correlation of input streams. In our target runtime system, the Stream Processing Core (SPC) (Jain *et al* 2006), a workflow is represented as a DAG, where vertices represent data sources and PEs, and edges represent data streams. Each PE exposes a number of input and output ports, where it can receive input streams and produce output streams. Each data stream arriving from an external source or produced by an output port of a PE, can be connected to one of the input ports of another PE, stored, or sent directly to the end users console. An example of a workflow is shown in Figure 2). This workflow is constructed in response to a user query for traffic congestion levels for a particular roadway intersection, e.g. Broadway and 42nd in New York City. A workflow constructed for such a query may use raw data from different sources. It may use video from a camera at the intersection, extracting images from the video stream and examining them for alignment to visual patterns of congestion at an intersection. In order to improve accuracy, it may also draw audio data from a sound sensor at the intersection and compare it with known congestion audio patterns. The end result is achieved by combining feeds from the two analytic chains. The figure also shows the format of the data packets exchanged between different components.

In our planning model, the state of the world consists of a set of streams. The initial state consists of streams produced by external sources. During planning, the state also includes new streams produced by PEs included in the workflow.

Each stream is described in terms of a typical packet it carries. The description of the packet is in terms of a set of OWL ABox assertions, represented as an RDF graph. For example, consider the stream produced by the video camera source in Figure 3. Every packet on this stream has two data elements: a video segment and a time interval, described as the exemplar individuals, _VideoSegment_1 and _TimeInterval_1. The constraints obeyed by these data ele-

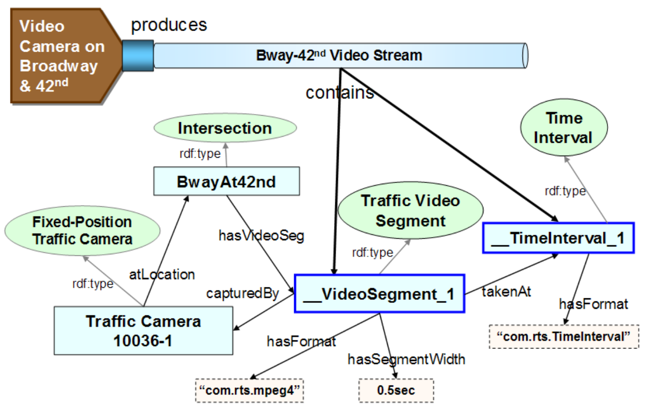

Figure 3: Description of a Video Camera Data Source. Ovals represent OWL classes, rectangles are OWL individuals, and dashed rectangles are literals

ments are described using an RDF graph. A specific packet on this stream would contain a specific video segment and a specific time interval, that obey these constraints. Note that this description uses classes, properties and individuals defined in one or more domain OWL ontologies.

Each action (PE) can have multiple input and multiple output ports. An action can be applied only in the state where for each input port there is an existing stream that matches the precondition specified on the port. Once an action is applied, it performs a state transition by creating new streams. Our model provides a blackbox description of the PE; it only describes the inputs and outputs, and doesn't model the internal state of the PE. The action is specified as a graph transformation operation, where the PE transforms an RDF graph pattern describing the input requirements to an RDF graph pattern describing the output stream.

For example, consider the VideoImageSampler in Figure 4, which has one input and one output. Any input stream connected to this PE must carry packets that contain two data elements: a video segment (?VideoSegment_1) and a time interval (?TimeInterval_1). The PE analyzes this input packet and produces as output an packet containing two new objects: an image (__Image_1) that it extracts from the video segment, and a time (__Time_1) for the image, which lies within the input time interval. There are other constraints associated with these data elements in the input and output packet, such as (?VideoSegment_1 takenAt ?TimeInterval_1), and (?VideoSegment_1 hasSegmentWidth 0.5sec).

A more detailed description of the model of streams and stream processing components can be found in (Liu, Ranganathan, & Riabov 2007b).

Planner

As described earlier, our planning model is based on SPPL, and can hence employ many of the planning techniques developed for SPPL (Riabov & Liu 2006). However, the use of DLP reasoning to check action applicability requires the use of new techniques. Traditional approaches to combining reasoning with planning relied on the invocation of a DL reasoner to evaluate preconditions (e.g. (Sirin & Parsia 2004)). Combining DL reasoning with planning presents significant performance challenges, since the planning process may result in a large number of invocations to a DL reasoner, which can be expensive. Besides, using a reasoner during planning makes it difficult to reuse many of the optimizations that have been proposed for SPPL planning.

As a result of these challenges, we have developed a planner that avoids calling a DL reasoner during planning by using a two-phase planning approach. In the first phase, which occurs offline, the planner translates descriptions of actions into SPPL. During the translation phase, the generator also does DLP-reasoning on the effect descriptions to generate additional inferred facts about the effects. It uses the Minerva DLP reasoner (Zhou *et al*) for this purpose. The SPPL descriptions of different actions are persisted and re-used for multiple goals. The second phase is triggered whenever a composition is required. In this phase, the planner performs reasoning on the initial state to determine additional inferred facts about the initial state. It then generates a plan using an SPPL planner (Riabov & Liu 2006). No DL reasoner is invoked during this plan generation. Further details about the two-phase approach to planning can be found in (Liu, Ranganathan, & Riabov 2007a).

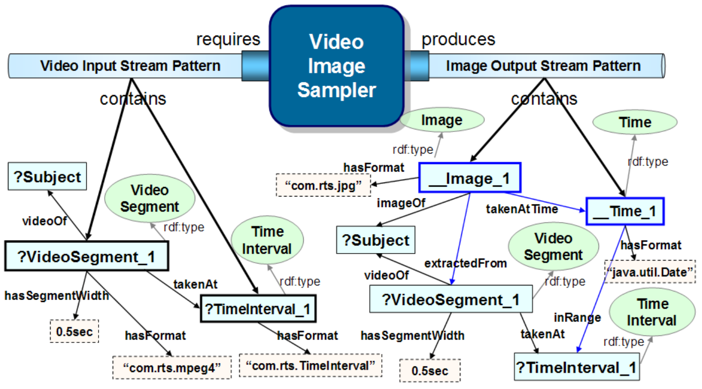

Figure 4: Description of VideoImageSampler PE

Domain and Problem Construction

One of the challenges facing AI planning is in promoting effective sharing and reuse of domain knowledge across different systems and applications. Domains represented in languages like PDDL are often constructed for a specific application and can be difficult to reuse. In this section, we describe some of the methodologies used in our system to facilitate the development and reuse of domains represented in OWL.

There are a number of advantages in using OWL as a basis for representing domains. Various tools are available for editing and validating OWL ontologies. OWL is represented using RDF, which uses URIs to identify resources and hence provides a standard, global naming mechanism for different terms. RDF is also inherently graph-based, which allows the use of graphical editing tools for authoring and visualizing RDF graphs. Finally, the import mechanism in OWL ontologies allows developing modular portions of the ontology that can be reused in different applications.

In order to represent action descriptions, we have developed a language called SGCDL (Semantic Graph based Component Description Language). The language describes the precondition and effect of actions using triples represented in an N3 format (Berners-Lee 1998). The language allows importing OWL ontologies and using terms defined in these ontologies for describing actions. We have developed a parser for this language that validates both the DL consistency and the planning requirements. The DL validation is done by using a DL reasoner (in our case, the Minerva DLP reasoner (Zhou *et al*)) and it ensures that the precondition and effect description is consistent with respect to the imported ontologies. The planning requirements are checked through a set of rules which ensure that the descriptions are valid for planning. An example of a rule is that no free variables appear in the effect descriptions.

We have developed a collaborative domain construction framework, where people of different roles can contribute to the domain (Figure 5). Domain experts develop or reuse OWL ontologies relevant for any domain. The descriptions of PEs are provided by "action specifiers", who encode descriptions of actions. In the workflow composition use-cases, these action specifiers are developers of processing components can also create semantic descriptions of the components in SGCDL.

The different ontologies are described in a modular fashion to enable high reuse. High-level ontologies (describing generic concepts like location and time) are used by different domains. There are also a number of domain-specific ontologies. Descriptions of actions can import any of the ontologies. We use a source control system (Rational ClearCase), that supports versioning and branching, to allow users to create and edit OWL and SGCDL files. Users can check out and use tools like Protégé for editing the ontologies. The source control is at file-level. We are investigating other source control systems that allow control at a finer granularity such as graph-level or triple-level.

In order to aid specification of the domain, our framework provides diagnostic tools that allow developers to check if their action specifications are compatible with existing action specifications. For example, the tools help determine if two actions can be placed in a sequence in a plan under different conditions. These features are particularly useful for debugging the descriptions of actions.

Conclusion

In this paper, we have presented a planning and knowledge engineering framework based on OWL ontologies that facilitates the development of domains and the use of description logic reasoning while planning. While the framework has been applied towards composing workflows in stream processing systems, it is quite general and can be applied in other domains.

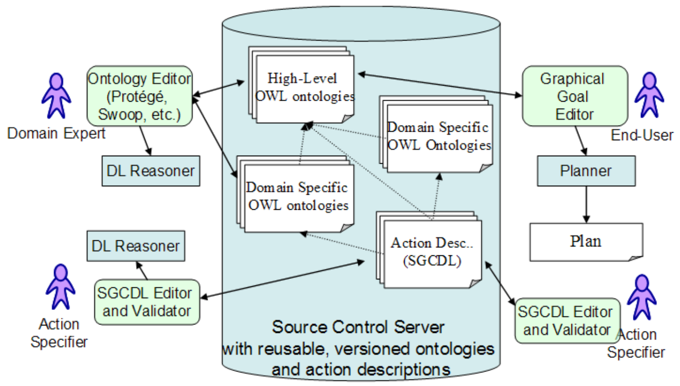
Domain Expert
Ontology Editor (Protégé, Swoop, etc.)
DL Reasoner
High-Level OWL ontologies
Domain Specific OWL Ontologies
Domain Specific OWL ontologies
Action Desc.. (SGCDL)
Graphical Goal Editor
End-User
Planner
Plan
DL Reasoner
SGCDL Editor and Validator
Action Specifier
SGCDL Editor and Validator
Action Specifier
Source Control Server with reusable, versioned ontologies and action descriptions

Figure 5: Knowledge Engineering framework showing different people accessing shared domain description

References

Baresi, L., and Heckel, R. 2002. Tutorial introduction to graph transformation: A software engineering perspective. In *1st Int. Conference on Graph Transformation*.

Beckett, D. 2004. Rdf/xml syntax specification. *http://www.w3.org/TR/rdf-syntax-grammar*.

Berners-Lee, T. 1998. Notation 3. Technical report, World Wide Web Consortium (W3C).

Grosof, B.; Horrocks, I.; Volz, R.; and Decker, S. 2003. Description logic programs: combining logic programs with description logic. In *WWW*.

Jain *et al*, N. 2006. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In *SIGMOD'06*.

Liu, Z.; Ranganathan, A.; and Riabov, A. 2007a. A planning approach for message-oriented semantic web service composition. In *AAAI*.

Liu, Z.; Ranganathan, A.; and Riabov, A. 2007b. Use of owl for describing stream processing components to enable automatic composition. In *OWLED*.

McGuinness, D., and van Harmelen, F. 2004. OWL web ontology language overview. In *W3C Recommendation*.

National cancer institute thesaurus. *http://www.mindswap.org/2003/CancerOntology/*.

Noy, N., and Rector, A. 2004. Defining n-ary relations on the Semantic Web. `http://www.w3.org/TR/swbp-n-aryRelations/`

Rector, A. L., and Horrocks, I. R. 1997. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *AAAI*.

Riabov, A., and Liu, Z. 2006. Scalable planning for distributed stream processing systems. In *ICAPS'06*.

Rozenberg, G. 1997. *Handbook of Graph Grammars and Computing by Graph Transformation, vol 1*.

Sirin, E., and Parsia, B. 2004. Planning for Semantic Web Services. In *Semantic Web Services Workshop at 3rd ISWC*.

Zhou *et al*, J. Minerva: A scalable OWL ontology storage and inference system. In *1st Asian Semantic Web Symp. '04*.