InfraEngine: Inferencing in the Semantic Web by Planning

Christoph Wernhard
info@cs.christophwernhard.com

– Draft 2002, revised 2007 –

Abstract. The idea of InfraEngine is to help establishing a Semantic Web infrastructure by a specially adapted AI planning engine. Inputs are distributed Web documents in Semantic Web formats proposed by the World Wide Web Consortium. Outputs are also delivered in such formats. The user interface allows to browse Semantic Web data and to control the planning services from any Web browser. Also other programs can make use of these services. A small and understandable, but general, mechanism that can be used for different kinds of applications should be provided.

1 Introduction

1.1 Preliminaries: Resource Oriented View of Planning

States and Actions. Executing an *action* leads from a *state* of the world to a successor state. A state is represented by a multiset[1] of *fluents*[2].

Planning Rules. The effects of an action are specified by a set of *planning rules* for that action. A planning rule has two parts, its *antecedent* and its *consequent*. Both parts are multisets of fluents.

For example, a rule for an action `buy(Object, Seller, Price)`:[3]

```
  [ have(Object), budget(Budget - Price) ] <--
           [ forsale(Object, Seller, Price), budget(Budget) ].
```

Operational Reading of Rules. The fluents in the antecedent are matched against the fluents in the initial state: for each fluent in the antecedent, a matching fluent in the state is removed. The fluents in the consequent are then added to the state, to yield the successor state. Matching is performed using *unification* (see e.g. [Nil98]): rules and states may contain *logic variables*, which are bound during matching.

[1] A *multiset* is similar to a set, except that with each element a number of occurrences is considered. Multisets are also known as *bags*.

[2] I.e. facts, whose truth value is state dependent.

[3] Fluents are written here in a Prolog like notation. Capitalized symbols are logic variables.

Fluents are considered as resources, objects that can be consumed and provided. A rule application means that the fluents of the consequent are provided, after the fluents in the antecedent have been consumed.

Applying the example rule to the following state:

```
[ forsale(book1, shop1, 30),
  forsale(book2, shop1, 35),
  budget(40) ]
```

yields the successor state:

```
[ have(book1), forsale(book2, shop1, 35), budget(10) ].
```

Planning Tasks and Plans. Given a set of rules, a start state and a goal state, a *planning task* is to determine, whether there exists a sequence of rule applications that leads from the start to the goal state.

A *plan* is the proof graph of such a solution. Its nodes are labeled by actions. An action is represented by a term that may share variables with its rules. The plan represents a partial ordering of action instances[4]. Any sequence of action applications compatible with the partial ordering leads from the start to the goal state.

In the example, the plan would consist of a single instance of action `buy(book1, shop1, 30)`. Executing this action means buying `book1` from seller `shop1` for the price of 30 dollars.

Situation Calculus. This resource oriented formulation of planning is equivalent to the *situation calculus*, which was the original formulation of planning [MH69,HS89] and is used as basis for many transition system languages, for example the PSL process specification language by NIST [SGCL00].

1.2 Ideas

Rules in the Web. Planning rules can be published in the Web like HTML documents. Consumed fluents in antecedents connect to produced fluents in consequents, somewhat similar to links.

Schemas in the Web. Schemas, i.e. class definitions, which may relate to each other by subclassing, can be published in the Web like HTML documents. By referring to equivalent schemas, it is ensured that different planning rules are about the same objects. Starting from schemas, application relevant rules can be found by humans or programs.

Planning Engine. An inference engine combines Web published rules and reasons about them. It can compute plans that answer queries how to reach a specified goal state from a specified start state.

[4] I.e. nodes labeled with actions.

The Semantic Web. Similar ideas for a world wide knowledge base have been in the air since the late nineties [San96,BL98]. The World Wide Web Consortium promotes this under the name *Semantic Web Activity*[5]. Although the seven layers of the Semantic Web architecture diagram includes layers *Logic* and *Proof*, up to now most of the work is focused on the lower *Ontology* layer, i.e. on schema languages and schemas. With RDF [LS99,Bec02,Hay02] and RDFS [BG02], the W3C published recommendations and drafts for a data exchange and a schema language for the Semantic Web. Another proposed Semantic Web schema language is DAML[6], which adds to RDFS a variant of description logic.

InfraEngine. The idea of InfraEngine is to help establishing a Semantic Web infrastructure by a specially adapted AI planning engine. Inputs are distributed Web documents that use Semantic Web formats proposed by the World Wide Web Consortium. Outputs are also delivered in these formats. The user interface allows to browse Semantic Web data and to control the planning services from any Web browser. Also other programs can make use of these services.

A small and understandable, but general, mechanism that can be used for different kinds of applications, should be provided.

1.3 Application Fields

Product Composition. Planning rules can be used to express composition of physical components, as illustrated by the following rule for action `assemble_bike`:

```
[ available(bike) ] <--
    [ available(wheel), available(wheel), available(frame) ].
```

So an electronic marketplace that includes the composition of products and services from different offerers to achieve a goal product could be a prototypical application. Composition rules can be given by both parties, the providers and consumers.

Intranet Knowledge Management. Another application field is Intranet knowledge management in large organizations. Distributed rules can express relationships that are orthogonal to the structuring of the organization, for example engineering or project management dependencies. A global management perspective is provided. Engineers can see what global effects local changes might have.

Reasoning about Processes. In contrast to "business processes" implemented without logic techniques, processes specified by planning rules can not be just executed, but also be reasoned about. The effects that different possible actions and courses of action would have can be compared. Other properties of them, for example the involved persons, can be computed in advance.

[5] `http://www.w3c.org/2001/sw/`

[6] `http://www.daml.org`

Related Application Fields. In general, planning rules can be used to express tasks like planning, scheduling, manufacturing resource planning, product data management, configuration management, workflow management and simulation. The Web embedding makes it possible to execute such tasks in a distributed manner. This openness is useful in application scenarios that benefit from the possibility of chaining several distributed pieces of information into a global view.

Reusability of Rules. Knowledge encoded as planning rules does not depend on some particular language or software system. The straightforward situation calculus semantics makes the rules highly reusable in various contexts.

A Difference from Expert System Shells. From the user's perspective, the Semantic Web comes already equipped with a large "world wide" knowledge base. This is in contrast to the expert system shells of the 1980s, which had to be laboriously filled to meet the particular application requirements.

2 The InfraEngine System

2.1 Overview

All in- and output data, such as queries, rules and schemas are RDF Web documents, respectively facts, that are maintained by the *Knowledge Base Management Unit*. The three further program components are the *Presenter*, which acts as a Web server and refines RDF facts obtained directly from documents or as output from the *Planner* to HTML for human users, or to or other formats such as RDF for machine clients. The *Composer* composes planning tasks from Web documents. The Planner, finally, is in essence a classic AI planning engine. A prototype implementation of the system is available from

`http://www.infraengine.com`.

2.2 Data

Data and Document Types. RDF is used as format for all in- and output data. Appropriate schemas are specified in RDFS. Data type specific document types, can thus be generated from the RDFS schemas, e.g. as XML DTDs.

Documents. All input data is considered as "residing" in RDF documents. A document is identified by an URI. An author might be associated with a document. A document might change its content over time.

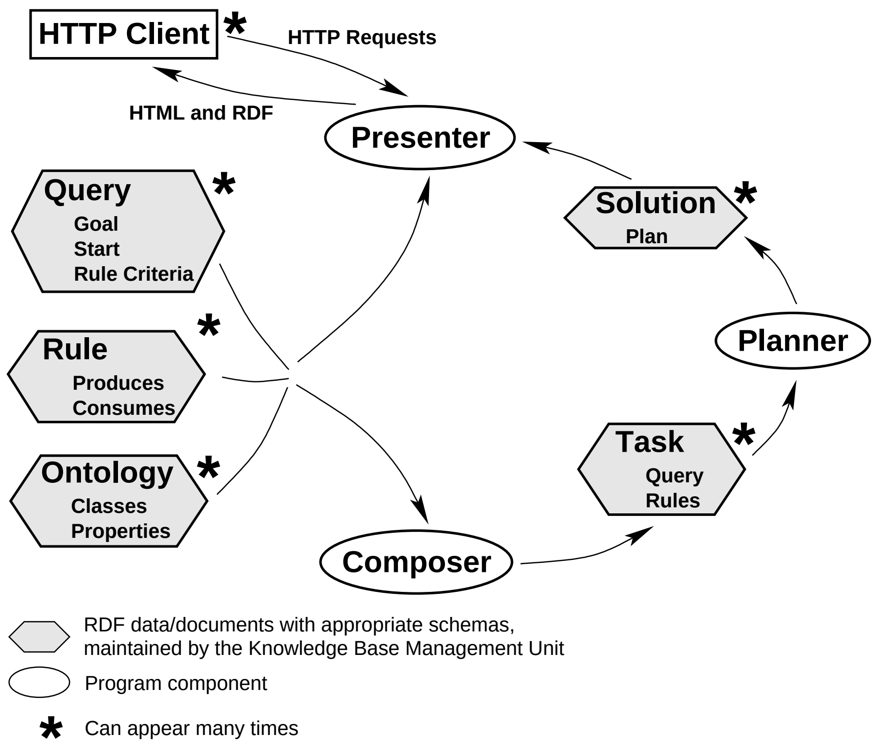

Fig. 1. InfraEngine System Architecture

Knowledge Bases. Different documents are combined to a *knowledge base*. Knowledge bases are maintained by the *knowledge base management unit*, which caches RDF facts and supports relational queries as well as inference tasks, such as computing RDF Model Theory closures[Hay02]. With a knowledge base, access rights for different users and user groups can be associated. A knowledge base can be refreshed, which means that its documents, if modified, are reloaded. Ad hoc constructed knowledge bases are used, e.g. to represent the solution of planning tasks. It is always possible to convert a knowledge base to an RDF document.

2.3 The Presenter

The Presenter is the user interface of the InfraEngine. It appears as an RDF browser that is enhanced by special views for planning solutions and commands to execute planning queries. It works as a Web server and can be run as personal user agent, for shared use in a work group or for general public use.

Through the Presenter, the InfraEngine can be accessed by Web browsers that receive generated pages in HTML and other formats such as RDF and XML DTDs. The complete functionality is also available via form URIs to machine clients. The HTTP user authentication mechanism is used to pass user name and password information.

Views. For RDF objects of certain types, such as classes, properties, tasks, solutions, specialized page types, so called *views* are available. From each view there is a direct link to all the other applicable views. A standard *object* view is available for all RDF objects. It shows the values of its properties and inverse properties. Graph views are available that show a clickable portion of the RDF graph surrounding the object.

Commands and TOC Views. Commands and queries can be started with command HTML form pages. "Table of content" views include the set of classes, set of properties and knowledge bases.

Abbreviations. As world wide identifiers, URIs can be quite long and unreadable. In the XML world a bunch of techniques has to be applied to cope with this: namespaces, entities, base URIs. The Presenter uses uniform abbreviations of URI prefixes. They are applied on the presentation level, internally the full "canonical" forms are used. However information about used abbreviations can be stored as meta information in knowledge bases. If appropriate, these abbreviations are mapped to XML namespaces in input and output documents.

Weaving with the Web and Natural Language. The Presenter can act as a Web proxy to browse arbitrary web pages. If these link to RDF documents, the proxy decorates the page with a menu to gather these pages into a knowledge base.

RDFS specifies a property `rdfs:comment` to associate comments with RDF objects. All views display these comments, if appropriate in an abbreviated form, along with RDF object identities (URIs).

2.4 The Composer

The Composer generates planning tasks, based on a given query. It can make use of abstracted meta information about rules, which can be obtained with the help of classic syntactic search engines.

2.5 The Planner

The core of the planning component is a classical AI planner that operates in backward chaining mode.

Planning Algorithm. The planning algorithm used is *linear backward chaining (LBC)* [Fro96]. The implementation is in Prolog and follows the PTTP approach [Sti88]. A planning problem is compiled into a Prolog program. The core planning system was tested with the the tasks of the AIPS 98 Planning Competition[7] and showed a performance comparable to the other planning systems in the competition.

[7] `ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html`

Constraint Handling. Constraint handling allows to embedded specialized inference mechanisms that are orthogonal to the overall proving process. If not enough information is obtained to decide a constraint during the proving process, the unsolved constraint can just be returned as part of the solution. Constraints can be used e.g. for arithmetic relationships or type handling. The planner implementation has a small interface that allows to hook in arbitrary constraint handlers.

2.6 Mapping Between RDF and Fluents

At first sight, a logic reading of RDF as binary facts seems straightforward, a closer look leads to many subtleties [Hay02,CK01]. In our approach to planning, fluents are never asserted as facts, but treated as objects from a logic point of view. This means they should be treated as terms, quoted facts, in RDF.

High flexibility for applications is achieved by allowing arbitrary RDF objects to appear as fluents and actions. Binary RDF facts are a special case by using objects of the class `rdfs:Statement`.

The type system that is currently supported internally is targeted at efficient processing with logic programming techniques. With it certain restrictions hold for fluents and actions: an object can only have a single value for each property, and an object must have a unique most special type. These restrictions must hold for the classes of actions and fluents as well as their their subobjects *as far as they are used* within a planning task.

A preprocessing step performs type inference using the RDF Model Theory closure operators on the type related information in the planning task and the RDFS schema.

3 Issues

3.1 Planning Method

- Special kinds of subsumption
  - Subsumption of solutions
  - Subsumption considering cost and time constraints
  - Subsumption that considers "decomposable" fluents, e.g.
    `have(X), have_qty(X, N-1) <-- have_qty(X, N)`
- Ranking of solutions
- Optimal solutions, with respect to cost, time or plan size
- An algorithm that scales to large inputs, similar to datalog processing techniques

3.2 Planner Extensions

- Planning rules with disjunctive heads
- Aggregation, e.g. `setof`
- Some form of negation
- A constraint system that supports a generally useful set of different constraint types

3.3 Further Reasoning Modes

Besides planning, there are other reasoning modes that might be useful for applications.

Projection. Determining the result state of applying a given plan to a given start state.

Postdiction. A form of abduction. Given are rules, a goal state and an incompletely specified start state. Computed are fluents that would be additionally required in the start state to reach the goal.

Scheduling. This is similar to planning, but the focus is on assignment of temporal information to actions.

Plan Execution, Grounding. Some types of actions can be directly executed electronically, for example making an order by e-mail or document routing tasks. Executions steps might be interleaved with the planning process ("reactive planning").

Reasoning about Process Properties. E.g. reachability and possibility of deadlocks.

3.4 Ontologies

Basic Ontologies, e.g. for business scenarios are required. These should be simple or adequate in the sense that a simple core part of an ontology can be easily used for simple applications.

3.5 Representation Issues

Application Patterns, Ramification. It seems that some applications such as the composition of physical objects (as in the examples given above) or selling acts can be straightforwardly represented by planning rules. However some aspects can be hard to express, for example a constraint that an object physically contained in another one, changes its location, when that of the container changes. Perhaps "application patterns" for planning should be worked out.

For some of the representation related difficulties, such as the qualification and ramification problem, there exists a number of solution approaches. Ramification in connection with backward chaining planners seems hardly considered in the literature.

Handling of Rule Sets. Perhaps a language level above the concrete planning rules is required: It may be required to systematically add or remove some aspects from a whole set of rules. For example parameters like actors or time and cost information. It may be useful to generate planning rules from information that is maintained as a plain fact base, for example a graph expressing some dependency relationship.

4 Related Work (as of 2002)

4.1 DAML Services

The DAML Services branch of the DAML program has the objective of developing a DAML-based Web Service Ontology (DAML-S), as well as supporting tools and agent technology to enable automation of services on the Semantic Web[Coa02].

Since services in DAML-S are modeled as processes, it should in principle be possible to use the InfraEngine as reasoner for DAML-S, but working this out may be not trivial. Semantics of DAML-S is specified by Petri-nets of a certain kind instead of the situation calculus. Perhaps disjunctive planning rules are required. Also the InfraEngine reasoner has not yet been adapted to process DAML schemas and objects.

4.2 RuleML

RuleML [BTW01] is an ongoing effort to create a taxonomy of rule based systems and make them accessible to the Semantic Web world. In the current version, the top node of the hierarchy is "reaction rules", which roughly correspond to planning rules but miss the clear situation calculus semantics. Reaction rules can only be used in forward chaining mode. Other logic systems are considered as specializations of reaction rules, which seems motivated by the idea of characterizing them by proof theoretic means (i.e. implementing them) in terms of the reaction rules.

References

[Bec02] Dave Beckett. RDF/XML syntax specification (revised), 2002. W3C Working Draft.

[BG02] Dan Brickley and R.V. Guha. RDF vocabulary description language 1.0: RDF schema, 2002. W3C Working Draft.

[BL98] Tim Berners-Lee. Semantic web road map, 1998. Draft.

[BTW01] H. Boley, S. Tabet, and G. Wagner. Design rationale of RuleML: A markup language for Semantic Web rules. In *International Semantic Web Working Symposium (SWWS)*, pages 384–401, Stanford, 2001.

[CK01] Wolfram Conen and Reinhold Klapsing. Logical interpretations of RDFS - a compatibility guide — working paper, 2001.

[Coa02] The DAML Services Coalition. DAML-S: Web service description for the semantic web. In *The First International Semantic Web Conference (ISWC)*, 2002.

[Fro96] Bertram Fronhöfer. Cutting connections in linear connection proofs. In *Int. Computer Symposium '96*, pages 109–117, December 1996.

[Hay02] Patrick Hayes. RDF model theory, 2002. W3C Working Draft.

[HS89] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. In D. Metzing, editor, *GWAI-89: Proc. of the 13th German Workshop on Artificial Intelligence, Eringerfeld, Deutschland*, pages 63–73. Springer, Berlin, Heidelberg, 1989.

[LS99] Ora Lassila and Ralph R. Swick. Resource description framework (RDF) model and syntax specification, 1999. W3C Recommendation.

[MH69] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*. Edinburgh University Press, 1969.

[Nil98] N. J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, Inc., San Francisco, 1 edition, 1998.

[San96] Erik Sandewall. Towards a world-wide knowledge base. In Maciej Michalewicz Zbigniew W. Ras, editor, *Foundations of Intelligent Systems, 9th International Symposium (ISMIS)*, volume 1079 of *LNAI*, pages 50–55, Berlin, 1996. Springer.

[SGCL00] C. Schlenoff, M. Gruninger, M. Ciocoiu, and J. Lee. The process specification language (PSL) overview and version 1.0 specification. Technical report, National Institute of Standards and Technology, Gaithersburg, 2000.

[Sti88] Mark E. Stickel. A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.