

An overview of SPARQLent

Marco Luca Sbodio*

September 28, 2009

Abstract

We present a brief overview of SPARQLent, a goal-directed intelligent agent with service matchmaking and planning capabilities. The functioning of SPARQLent is based on a novel approach that uses SPARQL to describe preconditions and postconditions of services.

1 Functioning

We use SPARQL as a formal language to describe the preconditions and postconditions of services, as well as the goal of intelligent agents. SPARQL query evaluation is used to check the truth of preconditions in a given context, and to build the description of the postconditions resulting from the service execution in that context.

We use a SPARQL **CONSTRUCT** query to give a compact representation of the preconditions and postconditions of a service. A **CONSTRUCT** query is made of a *template pattern* ε , and a **WHERE** clause consisting of a graph pattern π . The **CONSTRUCT** query yields an RDF graph G_ε formed by taking each query solution, substituting for the variables into the graph template ε , and combining the triples into a single RDF graph by set union. We use the graph pattern π to encode the preconditions of a service, and the template pattern ε to define its postconditions.

A SPARQLent is a goal-directed intelligent agent equipped with a knowledge base and a reasoning engine. The knowledge base is essentially an RDF graph, whose interpretation may differ depending on the entailment regime supported by the reasoning engine. The SPARQLent goal is given as a SPARQL **ASK** query, whose graph pattern represents a set of constraints defining the state of affair that the SPARQLent wants to bring about in the world.

The content of the SPARQLent knowledge base is interpreted as an (incomplete) description of the world state, and the services available over the Web are interpreted as actions whose execution cause a world state transition (similarly to Propositional Dynamic Logic). The SPARQLent discovery algorithm identifies those services whose preconditions are satisfied by the content of its knowledge base, and whose postconditions satisfy its goal. Intuitively, the discovery operation works as follows:

*Marco Luca Sbodio works at Hewlett-Packard Italy Innovation Center, and he is a PhD candidate at Université de Technologie de Compiègne (France) and Politecnico di Torino (Italy)

- a service is executable if and only if the graph pattern π representing its preconditions have solutions over the RDF graph G_a corresponding to the SPARQLent knowledge base; G_a is the (incomplete) description of the initial world ω_a ;
- if a service is executable, then the SPARQLent can compute the description of the service postconditions by executing the corresponding **CONSTRUCT** query over its knowledge base G_a ; let G_ε be the resulting RDF graph;
- the SPARQLent merges G_ε with G_a , and the resulting RDF graph G'_a is the description of the world state ω'_a resulting from the execution of the service in the initial world ω_a ;
- the SPARQLent checks the consistency of G'_a ; this operation depends on the entailment regime supported by the reasoning service, and may require special handling of inconsistent assertions (this an instance of the well known beliefs revision problem);
- after assessing (or enforcing) the consistency of G'_a , the SPARQLent verifies if its goal is satisfied by executing the corresponding **ASK** query over G'_a : if the query returns *true*, then the service execution has caused a transition to a world state where the SPARQLent goal is satisfied, and therefore the service is selected; otherwise the service is discarded.

We observe that the semantics of the service inputs (outputs) can be represented as additional constraints in its preconditions (postconditions): this is done by using triple patterns such as `_:x rdf:type foaf:Person`.

An additional feature of SPARQLent is the ability of progressively relax services preconditions. Thus a SPARQLent can also perform approximate match-making by discovering services whose preconditions are only partially satisfied by its knowledge base. This is done by identifying the unsatisfiable constraints of the graph pattern π representing the service preconditions, and by refactoring them to an optional subgraph. The relaxation algorithm ensures that a minimal number of constraints is relaxed to achieve preconditions satisfiability.

Finally, the SPARQLent planning capabilities are based on an algorithm that recursively discover services and combines them in sequences. It is essentially a form of regression planning.

2 SPARQLent adaptation for S3

The service descriptions provided by OWLS-TC, the test collection used for S3, do not contain formal definition of preconditions and postconditions. In this case our SPARQLent can only reason about service inputs/outputs and the corresponding subsumption hierarchies. Our SPARQLent plug-in for SME² provides six variants of SPARQLent, which use different combinations of entailment regimes, and approximate matchmaking capabilities (note that the planning capabilities are not used in SPARQLent plug-in for SME²).