A RESTful SWRL Rule Editor

Carsten Keßler

Institute for Geoinformatics, University of Münster, Germany
carsten.kessler@uni-muenster.de

Abstract. The sparse application of the Semantic Web Rule Language is partly caused by a lack of intuitive rule editors. This applies both from a human user's, as well as from a software interoperability perspective, as creating and modifying rules is currently hard in distributed Web applications. We introduce a Representational State Transfer-based approach that enables online rule editing to overcome these problems.

1 Introduction and Motivation

Rule-based reasoning still plays a minor role on the Semantic Web, despite the gain in expressivity[1] offered by the Semantic Web Rule Language (SWRL) [2]. While rules provide powerful solutions for problems that cannot be solved with standard Description Logic-based reasoning [3], the verbose syntax and a lack of intuitive user interfaces for rule editing hamper their application. Moreover, proprietary application programming interfaces complicate the integration into standards-based systems. We propose a wrapper for rule engines based on the Representational State Transfer (REST) approach [4] to overcome these problems. REST offers standardized, straight-forward access to resources without unnecessary overhead. Masking the complexity of rule editing as a RESTful service makes rule-based reasoning available for a wide range of services. At the same time, rule functionality is made more accessible for human users, as developers can reduce the editing options to the subset of SWRL's expressivity required for a specific application. Changes to single atoms can be completed without confronting users with the (potentially lengthy) complete rule, and adapted results can be directly returned in the server response. Besides the mapping approach from REST URIs to rules, we discuss the architecture of this RESTful wrapper. Moreover, we demonstrate its application in a mobile tool for personalized information retrieval that uses rules to represent user preferences.

2 Mapping REST to SWRL

Rule editing comprises creating, updating and deleting, in addition to simple access to the rules. These activities apply both for complete rules as well as for single atoms in the rules' bodies or heads. REST makes use of the `HTTP` request

[1] For OWL 2, the additional expressivity is partly covered in the RL profile [1].

methods GET, POST, DELETE and PUT to represent these activities. Resources are represented by descriptive URIs and can be queried and modified using these methods. Table 1 shows the required URIs for a complete rule editor.

Table 1. URIs for rule editing. The last five URIs also apply for head atoms.

Resource URI	HTTP Method	Description
/rules	GET	Lists all rule IDs
/rules	POST	Creates empty rule, returns ID
/rules/{id}	GET	Returns the rule with this ID
/rules/{id}	DELETE	Deletes the specified rule
/rules/{id}/body	GET	Lists IDs of all atoms in this rule's body
/rules/{id}/body	POST	Adds new atom to body, returns ID
/rules/{id}/body/{id}	GET	Returns a specific atom
/rules/{id}/body/{id}	PUT	Overwrites a specific atom
/rules/{id}/body/{id}	DELETE	Deletes a specific atom

Figure 1 gives an overview of the wrapper architecture. It creates a transparent access point for the KnowledgeBaseManager, providing access to the ontology repositories building the application knowledge base, and the RuleEngine. In case of the prototype implementation, Protégé Core has been used for knowledge base management, and Jess for rule execution[2]. The rule engine maintains the connection to the ontology and exposes its concepts and relations to the wrapper, which maps them to their respective URIs. Clients can modify rules via these URIs (see Table 1). When existing rules are modified, the (application-dependent) server response contains the knowledge inferred after rule execution. The rule engine also checks rules for validity before execution; if errors occur due to faulty client input, these must be passed on to the wrapper and forwarded to the client with the respective HTTP status codes.

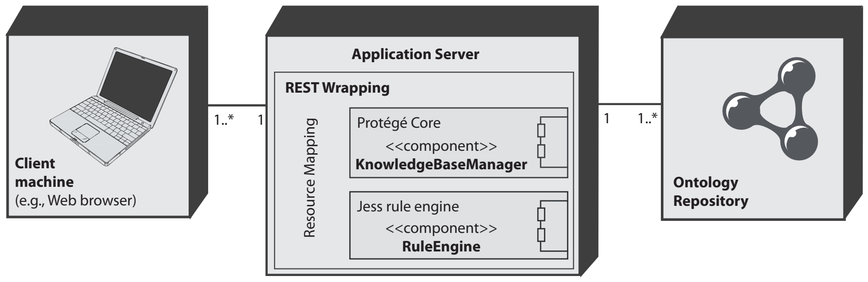
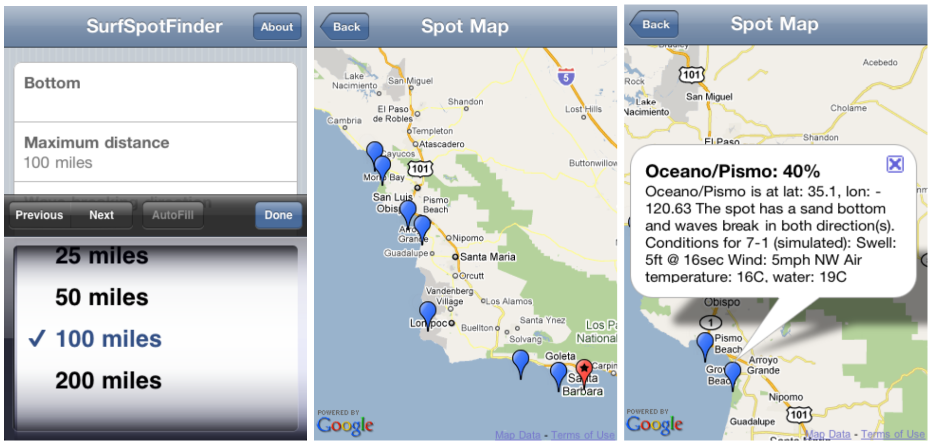


Fig. 1. Deployment diagram for the RESTful wrapper; adapted from [5].

[2] See http://protege.stanford.edu and http://jessrules.com.

3 Prototype: The Surf Spot Finder

The RESTful approach for SWRL rule editing described in Section 2 has been tested in the Surf Spot Finder [6], a mobile Web application for personalized recommendations for surf spots at California's central coast. The Web interface (see Figure 2) allows users to set their preferences for different aspects, each of which is mapped to one atom in the rule representing the user's profile. Upon rule execution, all surf spots instances in the knowledge base that match the user preferences are reclassified as appropriate for this user. In order to maintain the link of a user to her specific rule, the user's ID is stored in a cookie on the client side. On the server side, a new resource is created for every user, based on her ID (see Table 2). This maintains the stateless nature of REST, as no session data are stored, yet it allows users to preserve their setting between uses.



Bottom

Maximum distance
100 miles

25 miles
50 miles
✓ 100 miles
200 miles

Oceano/Pismo: 40%
Oceano/Pismo is at lat: 35.1, lon: -120.63 The spot has a sand bottom and waves break in both direction(s). Conditions for 7-1 (simulated): Swell: 5ft @ 16sec Wind: 5mph NW Air temperature: 16C, water: 19C

Fig. 2. Screen shots of the prototype user interface [5].

Table 2. Resources offered by the Surf Spot Finder prototype. The three operations for `bottom` are also available for all other aspects of the user profiles.

Resource URI	Method	Description
`/users`	`GET`	Returns a list of all users' rules
`/users`	`POST`	Creates new user, returns ID
`/users/{id}`	`GET`	Returns a user's SWRL rule
`/users/{id}`	`DELETE`	Deletes the specified user profile
`/users/{id}/spots`	`GET`	Returns matching spots
`/users/{id}/bottom`	`GET`	Returns the user's preferred bottom type
`/users/{id}/bottom`	`PUT`	Updates user's preferred bottom type, returns spots
`/users/{id}/bottom`	`DELETE`	Deletes user's preferred bottom type, returns spots

Imagine the following rule represents user 42's preferences: `SurfSpot(?spot)` $\wedge$ `hasBottom(?spot, 'rock')` $\rightarrow$ `Match(?spot)`. A `PUT` request to `http://somedomain.com/users/42/bottom` with `sand` as contents overwrites the corresponding atom, changing the rule to `SurfSpot(?spot)` $\wedge$ `hasBottom(?spot, 'sand')` $\rightarrow$ `Match(?spot)`. Execution of the rule reclassifies all matching SurfSpot instances as a Match, which are then returned to the client by the wrapper. Communication with the server component is asynchronous, allowing the application to update the map without reloading the whole application. When the user changes one of the parameters in her profile, this change is sent to the server and the updated set of matching spots is returned and shown on the map.

4 Conclusions and Future Work

We have proposed a RESTful Web service that enables online editing of SWRL rules. The general approach has been introduced and demonstrated in the Surf Spot Finder application that uses SWRL-based user profiles. The next steps in this research are the implementation of a *complete* mapping for rule editing as outlined in Section 2. Two aspects of this implementation are especially challenging. First, the automatic mapping of any built-ins, especially custom built-ins [7]. Second, an integrity checking for rule modifications that wraps potential error messages with the appropriate `HTTP` status codes.

Acknowledgments

This work has been funded by the German Research Foundation's *SimCat* project (DFG Ra1062/2-1 and Ja1709/2-2), see `http://sim-dl.sourceforge.net`.

References

1. W3C: OWL 2 Web Ontology Language – Profiles. W3C Recommendation, 27 October 2009: http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/ (2009)
2. W3C: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004: http://w3.org/Submission/2004/SUBM-SWRL-20040521/ (2004)
3. Horrocks, I.: OWL rules, OK? In: W3C Workshop on Rule Languages for Interoperability. (2005)
4. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, USA (2000)
5. Keßler, C.: Context-aware Semantics-based Information Retrieval. PhD thesis, Institute for Geoinformatics, University of Münster, Germany (May 2010)
6. Keßler, C., Raubal, M., Wosniok, C.: Semantic Rules for Context-Aware Geographical Information Retrieval. In Barnaghi, P., ed.: 4th European Conference on Smart Sensing and Context, EuroSSC 2009. Springer Lecture Notes in Computer Science 5741 (2009) 77–92
7. O'Connor, M., Das, A.: A Mechanism to Define and Execute SWRL Built-ins in Protégé-OWL. In: 9th Int. Protégé Conference, July 23–26, 2006, Stanford, California. (2006)