Composition of Linked Data-based RESTful Services

Steffen Stadtmüller

Institute of Applied Informatics and Formal Descriptions Methods (AIFB)
Karlsruhe Institute of Technology, Germany
Steffen.Stadtmueller@kit.edu

Abstract. We address the problem of developing a scaleable composition framework for Linked Data-based services, that retains the advantages of the loose coupling fostered by REST.

1 Problem Statement

The Linking Open Data community has gained momentum over the last years. At the same time there is a strong movement in the Web community toward a resourceful model of services based on Representational State Transfer (REST [3]) which propagates the primacy of loose coupling. Flexibility, adaptivity and robustness are direct consequences from the loose coupling and are achieved with links between resources, which allow clients to navigate from one resource to another during their interaction [11]. REST is particularly useful for software architectures in distributed data driven environments such as the Web [10].

Following the motivation to look beyond the exposure of fixed datasets, an extension of Linked Data with REST technologies has been proposed and explored for some time [1, 16].

The composition of RESTful resources originating from different providers suffers particularly from the necessary manual effort to use them. The reliance on natural language descriptions has led to mashup designs in which programmers are forced to write glue code with little or no automation and to manually consolidate and integrate the exchanged data.

Our contributions toward a scaleable loosely coupled composition will be

- an analysis of how self-descriptive resources have to be designed to enable composition;
- a service model for REST based on state transition systems as formal grounding for our composition;
- a declarative rule-based execution language to allow an intuitive specification of the interaction with resources from different providers;
- an execution engine as artifact to perform the defined interactions, which we want to evaluate with regard to scaleability.

The rest of the paper is structured as follows: In Section 2 we detail the existing work. In Section 3 we describe the methods with which we intend to leverage the advantages of Linked Data based REST architectures. We conclude in Section 4.

2 Related Work

Pautasso introduces an extension to BPEL [9] to allow a composition of REST and traditional web services.

There are several approaches that extend the existing WS-* stack with semantic capabilities by leveraging ontologies and rule-based descriptions (e.g., [14, 2, 6]). In contrast to WS-* are REST architectures build around another kind of abstraction: the resource. Therefore our approach is more focused on resource/data centric scenarios in distributed environments (e.g., in the Web).

RESTdesc [15] is an approach in which RESTful Linked Data resources are described in N3-Notation. The composition of resources is based on an N3 reasoner and stipulates manual interventions of users to decide which links should be followed.

Hernandez et al. [5] proposes a model for semantically enabled REST services as a combination of pi-calculus and an extension of triple space computing by Simperl et al. [12]. Similar to the idea of triple spaces is the composition of RESTful Linked Data resources in a process space, proposed by Krummenacher et al. [7] based on resources descriptions using graph patterns. Speiser and Harth [13] propose similar descriptions for Linked Data Services. Our approach shares the idea that graph pattern described resources read input from and write output to a shared space. We want to improve on this approach by providing a rigid service model and a more explicit way of defining the interaction with resources.

3 Methodology

In this section, we describe in more detail how we want to address the challenges we face in the development of a flexible and scalable composition framework.

3.1 Resource Descriptions

In a RESTful interaction with Linked Data resources only the HTTP methods can be applied to the resources. The semantics of the HTTP methods itself is defined by the IETF[1] and do not need to be explicitly described.

The state of Linked Data resources is expressed with RDF. It is sensible to serialise the input data, i.e., data that is submitted to resources to manipulate their state, in RDF as well. To convey the resulting state change after application of a HTTP method we use RDF output messages. In previous work [8] we analysed the potential of graph patterns, based on the syntax of SPARQL, to describe required input as well as their relation to output messages. The resulting graph pattern descriptions are attached to the resource. Therefore the resources stay self-descriptive.

[1] `http://www.ietf.org/rfc/rfc2616.txt`

3.2 REST Service Model

A REST service can be identified with the resources it exposes. An interaction within a REST architecture is based on the manipulation of the states of the exposed resources.

We develop a service model, that allows to formalise the functionalities exposed by a service based on Linked Data resources. The formal service model serves as rigid specification of how the use of individual HTTP methods influences resource states and how these state changes are conveyed to interacting clients.

We model a Linked Data-based RESTful service as a REST state transition system (RSTS). A state in the RSTS is defined as the set of states of all resources that are exposed by the service. The transitions between states are described with state change functions and output functions for every HTTP method respectively. The intuition behind the state change functions is that a state transition in the RSTS is effected by influencing resource states with HTTP methods. The intuition behind output functions is, that the application of an HTTP method on a resource also results in a defined output, that communicates the effected state change to the interacting client with an RDF message.

3.3 Execution Language

To allow programmers to formalise their desired interactions we develop a declarative rule-based execution language. The head of a rule corresponds to an update function of the RSTS in that they describe an HTTP method that is to be applied to a resource. The rule bodies are conjunctive queries that allow programmers to express their intention under which condition a method is to be applied. The use of conjunctive queries is motivated by the idea that clients have to maintain a knowledge space (KS) in which they store their knowledge about the states of the resources they interact with [7]. KS is filled with the RDF data the client receives after applying an HTTP method, as defined by the output functions of the RSTS.

We plan to develop an interpreter for our execution rule language as execution engine that can be integrated in applications. To achieve a fast scalable interpreter we plan to build the execution engine with a query engine based on the Rete algorithm [4], which allows a multithreaded, parallel evaluation of multiple queries.

We want to evaluate the performance of our engine with regard to (1) the amount of the communicated data, (2) the number of the composed services, (3) the complexity of the queries. We intent to implement several composition scenarios with a focus on real world services. We want to measure the execution time of the scenario implementations and compare the performance with implementations of the same scenarios based on standard SPARQL query engines,

with function mapping[2] for remote procedure calls, and other production rule engines (e.g., drools[3], JESS[4])

4 Conclusion

We have proposed to exploit the advantages resulting from the combination of REST architectures and Linked Data for a composition framework for REST services. We have scetched a declarative rule-based execution language with an with a state transition system as formal grounding and the challenges we address with this language, as well as an execution engine. For evaluation we intend to analyse real world scenarios build with existing services.

References

1. Berners-Lee, T.: Read-write linked data (Aug 2009), http://www.w3.org/DesignIssues/ReadWriteLinkedData.html
2. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer (2006)
3. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
4. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. AIJ 19(1), 17–37 (1982)
5. Hernández, A.G., García, M.N.M.: A formal definition of restful semantic web services. In: WS-REST. pp. 39–45 (2010)
6. Kopecky, J., Vitvar, T., Fensel, D.: Microwsmo: Semantic description of restful services. Tech. rep., WSMO Working Group (2008)
7. Krummenacher, R., Norton, B., Marte, A.: Towards Linked Open Services. In: FIS (2010)
8. Norton, B., Stadtmüller, S.: Scalable discovery of linked services. In: RED (2011)
9. Pautasso, C.: Restful web service composition with bpel for rest. DKE 68(9), 851–866 (2009)
10. Pautasso, C., Wilde, E.: Why is the web loosely coupled?: a multi-faceted metric for service design. In: WWW. pp. 911–920 (2009)
11. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (2007)
12. Simperl, E., Krummenacher, R., Nixon, L.: A coordination model for triplespace computing. In: COORDINATION (2007)
13. Speiser, S., Harth, A.: Integrating linked data and services with linked data services. In: ESWC (2011)
14. Studer, R., Grimm, S., Abecker, A. (eds.): Semantic Web Services: Concepts, Technologies, and Applications. Springer (2007)
15. Verborgh, R., Steiner, T., Deursen, D.V., de Walle, R.V., Valls, J.G.: Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc. In: NWeSP (2011)
16. Wilde, E.: Rest and rdf granularity (May 2009), http://dret.typepad.com/dretblog/2009/05/rest-and-rdf-granularity.html

[2] http://www.w3.org/TR/rdf-sparql-query/#FunctionMapping

[3] http://www.jboss.org/drools/

[4] http://www.jessrules.com/