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. Applications are increasingly focused on the use and manipulation of data resources distributed on the Web. Consequently REST gains popularity with its resource-centric interaction architecture and flexibility enabled by hypermedia controls, i.e., links between resources . The composition of RESTful services is often based on a manual ad-hoc development of mashups . The extension of Linked Data to allow for the RESTful manipulation of resources can bring new possibilities for composition approaches with a higher degree of automation. 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 with the trend towards opening up public sector and other data [2]. 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 [7]) which propagates the primacy of loose coupling. Flexibility, adaptivity and robustness are direct consequences from the loose coupling of REST and are particularly useful for software architectures in distributed data driven environments such as the Web [18].

REST defines the interaction between a client and a server as the manipulation of states of URI-identified resources with a constraint set of operations, i.e., the HTTP methods. Further, hypermedia controls (i.e., links to other resources) allow clients to navigate from one resource to another during their interaction [19]. The Linked Data design principles[1] also address the use of URI-identified resources and their interlinkage. However Linked Data is only concerned with the provisioning and retrieval of data. An extension of Linked Data with REST to allow for resource manipulation is therefore natural.

Following the motivation to look beyond the exposure of fixed datasets, the extension of Linked Data with REST technologies has been proposed and explored for some time [1, 27] and led recently to the establishment of the *Linked Data Platform*[2] W3C working group.

[1] `http://www.w3.org/DesignIssues/LinkedData.html`
[2] `http://www.w3.org/2012/ldp/charter`

Especially in data driven scenarios an increased value comes from the combination of data and the functionality to manipulate them. The increased value of such compositions is reflected in the constant growth of the mashup ecosystem of web APIs [26].

In a REST architecture, client and server are supposed to form a contract with content negotiation, not only on the data format but implicitly also on the semantics of the communicated data, i.e., an agreement on how the data have to be interpreted [25]. Since the agreement on the semantics is only implicit, programmers developing client applications have to manually gain a deep understanding of the provided data, often based on natural text descriptions of the services.

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.

On the other hand, traditional service composition approaches that aim to decrease the manual effort lead to a tight coupling between client and server, i.e., they sacrifice flexibility and are prone to failures due to server-side changes. Traditional composition approaches often fail to leverage hypermedia controls and do not provide straightforward mechanisms to dynamically react to state changes of resources. The reaction on state changes becomes especially important in a distributed environment, since a client can not ex ante predict the influence of other clients on the resources, i.e., REST does not allow a client to make assumptions on resource states.

Our hypothesis is that the combination of REST with Linked Data introduces the possibilities for a data and resource driven composition with a higher degree of automation, that preserves loose coupling by

- leveraging hypermedia controls provided by Linked Data resources
- specifying desired interactions dependend on resource states, which is enabled by a uniform state description format, i.e., RDF.

The development of a composition framework for REST is especially challenging, since the hypermedia controls and the resource states can only be determined during runtime, however, programmers have to specify their desired interactions at design time. A further requirement for our composition approach in a web based environment is a fast and scaleable execution of the composed services: A rapid interaction with resources from many different providers has to be possible to allow for the development of useable web based applications. Further, we want to cover a great variety of application and communication scenarios.

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.

Example 1. Lin et al. [14] propose a scenario where the search for information objects identifies additional actions that can be performed on the objects. E.g., the search for a movie presents actions such as reading reviews, adding it to a netflix queue and listening to the soundtrack. Our approach goes beyond the presentation of actions from different providers, and allows to develop clients that execute the actions in a specified dynamic manner: A client in which users can look for movies; if the review rating of an identified movie is above a threshold the movie is added to the users netflix queue and the soundtrack is played. Such clients can be applications such as apps for a handheld device or be deployed on the web themselves as encapsulated program resources.

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. In Section 4 we describe how we intend to evaluate our success. We conclude in Section 5.

2 Related Work

Pautasso introduces an extension to BPEL [17] to allow a composition of REST and traditional web services. To allow for a BPEL composition REST services are wrapped in WSDL descriptions.

There are several approaches that extend the existing WS-* stack with semantic capabilities by leveraging ontologies and rule-based descriptions (e.g., [22, 6, 4]) to achieve an increased degree of automation in high level tasks, such as service discovery, composition and mediation. Those approaches extending WS-* became known as Semantic Web Services (SWS). An Approach to combine RESTful services with SWS technologies in particular WSMO-Lite [24] was investigated by Kopecky et al. [11]. In contrast to SWS do REST architectures not allow to define arbitrary functions, but are constrained to a defined set of methods and are 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).

The scripting language S [3] allows to develop Web resources for REST interactions with a focus on performance due to parallelisation of calculations. In their definition resources can make use of other resources, thus also enabling a way of composing REST services. S does not explicitly address flexibility aspects of REST and has no explicit facilities to leverage hypermedia controls or to infer required operations from resource states.

RESTdesc [23] 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 hypermedia controls should be followed.

Hernandez et al. [10] proposes a model for semantically enabled REST services as a combination of pi-calculus [15] and approaches to triple space computing [5] pioneered by the Linda system [9]. They argue, that the resource states can be seen as triple spaces,

where during an interaction triple spaces can be created and destroyed as proposed in an extension of triple space computing by Simperl et al. [20].

Similar to the idea of triple spaces is the composition of RESTful Linked Data resources in a process space, proposed by Krummenacher et al. [12] based on resources described using graph patterns. Speiser and Harth [21] propose similar descriptions for RESTful 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. We address

- *resource descriptions* to allow to predict the effect of the execution of a functionality before invocation;
- a formal *service model* as grounding to describe the interactions that are offered and RESTful Linked Data resources, potentially spread over different servers;
- an *execution language* to instantiate a concrete interaction between a client and resources, which preserves the adaptability, robustness and flexibility of REST.

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[3] and do not need to be explicitly described.

The state of a 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 [16] we analysed the potential of graph patterns, based on the syntax of SPARQL[4], 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.

We want to analyse methods to serialise graph pattern descriptions and to attach them to the resources. Also, we have to decide how to tradeoff the necessary expressivity of the graph patterns and the resulting computational complexity. In the first steps we want to focus on basic graph patterns.

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.

[3] `http://www.ietf.org/rfc/rfc2616.txt`
[4] `http://www.w3.org/TR/rdf-sparql-query/#GraphPattern`

We want to develop a service model, that allows to formalise the functionalities exposed by a service based on Linked Data resources. A 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) similar to a state machine as defined by Lee and Varaiya [13]. The behavior of the clients themselves is not in the scope of this model, rather all possible interaction paths of a client with the resources are formalised.

A state in the RSTS is defined as the set of states of all resources that are (potentially) exposed by the service, serialised with RDF. Note that the set of resources can be infinite, since a service can allow to create additional resources. We consider the serialisation of a state of a resource that does not exist to be an empty set.

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. HTTP methods that do not change any resource states, describe self-transitions, i.e., transitions that start and end in the same state.

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 success with an HTTP status code. Further the output contains an RDF message that describes the the effected state change.

The possible interactions with the resources of several services can easily be formalised together in one RSTS as the side-by-side composition [13] of the transition systems of the individual services. Intuitively the side-by-side composition results in an RSTS, whose states contain the set of the states of the resources from both services.

The defined service model serves as formal grounding of the execution language described in Section 3.3. However, the self-descriptive resources are sufficient to define a RESTful services:

- The current state of Linked Data resources - and therefore the state of the RSTS - can be accessed as RDF.
- The possible transitions and the state they result in is declared with the graph pattern descriptions.

3.3 Execution Language

In a resource-driven environment we identify RESTful service composition with the interaction of a client with the resources of the services.

Definition 1. *The composition of two REST services S_1 and S_2 means the interaction with resources exposed by S_1 dependent on the resource states of S_2 and vice-versa. The dependency between the invoked state transitions (i.e., applied HTTP methods) and the states of resource is that*

1. *input data for the transition is derived from RDF detailing the states of resources and/or*

2. *the transition is only invoked if the resources are in a specified state.*

Therefore, to specify the interaction of a client with REST service resources and congruously the desired composition of the services, the state transitions that are to be invoked, have to be defined. Further the conditions, subject to the current states of resources, under which a specific transition is to be invoked have to be specified.

To allow programmers to formalise their desired interactions we intend to 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. Thus, programmers can define their desired paths through an RSTS with a set of rules for their client applications.

The use of conjunctive queries is motivated by the idea that clients have to maintain a knowledge space (KS) in which they stores their knowledge about the states of the resources he interacts with [12]. KS is filled with the RDF data the client receives after applying an HTTP method, as defined by the output functions of the RSTS. The output always informs the client about the current state after the application of the method.

Concretely we plan to use SPARQL queries, which are evaluated over KS. Queries are also used to dynamically, i.e., during runtime

- derive input data from the states of other resources, as stored in KS and
- identify the resource an HTTP method has to be applied to, i.e., leveraging hypermedia controls.

Further our execution language allows to define input and output of programs as graph patterns. The input pattern describes the structure of data that can initially be imported in KS to start the interaction as defined by the rules. After the interaction is completed the output pattern is evaluated over KS, thus extracting output data as result of the interaction. The notion of input and output allows to deploy the defined interaction itself (i.e., the composed service) as a resource in the Web, e.g., as servlet or cgi-bin. Note that the input pattern and output pattern are equivalent to the description pattern of other resources, which allows to encapsulate the interaction with resources and expose the functionality of a program.

We plan to develop an interpreter for our execution rule language as execution engine that can be integrated in applications. The engine implements the KS as well as the functionality to invoke an interaction with resources as defined with the execution language.

To achieve a fast scalable interpreter we plan to build the execution engine with a query engine based on the Rete algorithm [8], which allows a multithreaded, parallel evaluation of multiple queries.

To enable a wide variety of applications the engine will include an extension to support the interaction with REST services that are not based on Linked Data. The engine can store data entities (e.g., binaries, JSON documents) received from such services seperately. However, an interaction with such non-RDF entities requires to fall back to a more mashup-like composition.

4 Evaluation Plan

We want to evaluate the performance of our engine with regard to three influencing factors:

- the amount of the communicated data,
- the number of the composed services,
- the complexity of the queries.

We intent to implement several composition scenarios with a focus on real world services either developed by us or existing services wrapped to provide a Linked Data interface. To define relevant scenarios we identify domains in which we expect the influencing factors to be prevalent, e.g.,

- *geospatial services*: the gathering of data about places and areas combined with functionalities to visualise them, often leads to large amounts of data that have to be communicated;
- *social media services*: publishing information on many different social media platforms requires the interaction with resources from many different providers.
- *search services*: the search for information can be subject to complex conditions, which have to be coordinated by our engine.

We want to measure the execution time of the scenario implementations and compare the performance with implementations of the same scenarios based on

- mashup-style hard coded compositions,
- standard SPARQL query engines (e.g., ARQ[5]), with function mapping[6] for remote procedure calls,
- other production rule engines build on the RETE algorithm (e.g., drools[7], JESS[8])

Further we plan to qualitatively analyse the behavior of the different implementations when resources are changed or defective to evaluate flexibility and robustness.

5 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 a state transition system as formal grounding and the challenges we address with these language, i.e., achieving scalability and performance while preserving the flexibility and robustness of REST. Further we outlined an execution engine for the language. For evaluation we intend to analyse real world scenarios build with existing services.

[5] `http://jena.apache.org/documentation/query/index.html`
[6] `http://www.w3.org/TR/rdf-sparql-query/#FunctionMapping`
[7] `http://www.jboss.org/drools/`
[8] `http://www.jessrules.com/`

References

1. Berners-Lee, T.: Read-write linked data (Aug 2009), http://www.w3.org/DesignIssues/ReadWriteLinkedData.html
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. IJSWIS 5(3), 122 (2009)
3. Bonetta, D., Peternier, A., Pautasso, C., Binder, W.: S: a scripting language for high-performance restful web services. In: PPOPP (2012)
4. Cardoso, J., Sheth, A.: Semantic Web Services, Processes and Applications. Springer (2006)
5. Fensel, D.: Triple-space computing: Semantic web services based on persistent publication of information. In: INTELLCOMM. pp. 43–53 (2004)
6. 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)
7. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
8. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. AIJ 19(1), 17–37 (1982)
9. Gelernter, D.: Generative communication in linda. ACM TOPLAS 7, 80–112 (1985)
10. Hernández, A.G., García, M.N.M.: A formal definition of restful semantic web services. In: WS-REST. pp. 39–45 (2010)
11. Kopecky, J., Vitvar, T., Fensel, D.: Microwsmo: Semantic description of restful services. Tech. rep., WSMO Working Group (2008)
12. Krummenacher, R., Norton, B., Marte, A.: Towards Linked Open Services. In: FIS (2010)
13. Lee, E.A., Varaiya, P.: Structure and Interpretation of Signals and Systems. Addison-Wesley (2011)
14. Lin, T., Pantel, P., Gamon, M., Kannan, A., Fuxman, A.: Active objects: actions for entity-centric search. In: WWW. pp. 589–598 (2012)
15. Milner, R.: The polyadic pi-calculus. In: CONCUR (1992)
16. Norton, B., Stadtmüller, S.: Scalable discovery of linked services. In: RED (2011)
17. Pautasso, C.: Restful web service composition with bpel for rest. DKE 68(9), 851–866 (2009)
18. Pautasso, C., Wilde, E.: Why is the web loosely coupled?: a multi-faceted metric for service design. In: WWW. pp. 911–920 (2009)
19. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (2007)
20. Simperl, E., Krummenacher, R., Nixon, L.: A coordination model for triplespace computing. In: COORDINATION (2007)
21. Speiser, S., Harth, A.: Integrating linked data and services with linked data services. In: ESWC (2011)
22. Studer, R., Grimm, S., Abecker, A. (eds.): Semantic Web Services: Concepts, Technologies, and Applications. Springer (2007)
23. 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)
24. Vitvar, T., Kopecky, J., Zaremba, M., Fensel, D.: Wsmo-lite: Lightweight semantic descriptions for services on the web. In: ECOWS. pp. 77–86 (2007)
25. Webber, J.: REST in Practice: Hypermedia and Systems Architecture. O'Reilly (2010)
26. Weiss, M., Gangadharan, G.R.: Modeling the mashup ecosystem: structure and growth. RADMA 40(1), 40–49 (2010)
27. Wilde, E.: Rest and rdf granularity (May 2009), http://dret.typepad.com/dretblog/2009/05/rest-and-rdf-granularity.html