Proof-based Automated Web API Composition and Integration

Ruben Verborgh[1], Thomas Steiner[2], Erik Mannens[1], Rik Van de Walle[1], and Joaquim Gabarró Vallés[2]
[1] iMinds – Multimedia Lab – Ghent University, Belgium
{ruben.verborgh,rik.vandewalle}@ugent.be
[2] Universitat Politécnica de Catalunya – Department LSI, Spain
tsteiner@lsi.upc.edu

Abstract

Many providers offer Web APIs that expose their services to an ever increasing number of mobile and desktop applications. However, all interactions have to be explicitly programmed by humans. Automated composition of those Web APIs could make it considerably easier to integrate different services from different providers. In this paper, we therefore present an automated Web API composition method, based on theorem-proving principles. The method works with existing Semantic Web reasoners at a Web-scale performance. This makes proof-based composition a good choice for Web API integration. We envision this method for use in different fields, such as multimedia service and social service composition.

Keywords: service composition, Web API composition

1 Introduction

The number of publicly available Web APIs grows at a tremendous rate. By the end of 2012, more than 8,000 APIs are available, some of which are consulted billions of times per day [2]. Thanks to these Web APIs, mobile and desktop application developers can provide the functionality of many providers' services in their consumer applications. The provided services range from social activities *(e.g., share on Facebook or Twitter* over various detailed information supplies *(e.g., maps, events, or weather)*, to highly specific needs *(e.g., multimedia manipulation or language analysis)*. However, integrating these APIs into an application requires manual development work, such as writing the HTTP requests that need to be executed and parsing the returned HTTP responses. Instructions on how to write this code can often be found on the API's website in the form of human-readable API documentation.

Part of an automated solution to this problem is machine-readable documentation. On the lowest level, this documentation describes the message format and modalities; on a higher level, however, it also explains the specific functionality offered by the Web API, so a machine can autonomously decide whether the API is appropriate for a certain use case. In the past, we have proposed such a description method called RESTdesc, which offers an efficient way to capture the functionality of Web APIs [7,8] that is specifically tailored to REST or hypermedia APIs [6].

The other part of the solution is automated composition and integration, which is the application of machine-readable documentation to a specific problem. In other words, while programmers today need to interpret the human-readable documentation to choose an API and to implement it in an application, with automated integration, a machine would interpret machine-readable documentation, choose an API and interact with this API at runtime without human intervention. An important step in this direction is the automated matching and composition of Web APIs. In this paper, we therefore present an automated composition method for RESTdesc-described Web APIs, building upon our previous work.

2 Related work

RESTdesc is a Web API description method that captures the functionality of Web APIs, describing an HTTP request and its preconditions and postconditions [7,8]. RESTdesc descriptions are expressed in Notation3, which is a minor subset of RDF, the main Semantic Web language.

Composition of Web interfaces has mostly been discussed in the context of so-called "Big Web services", *i.e.*, services that do not conform to the REST principles [3], but rather employ HTTP as a tunneling protocol. Their composition, however, requires specific software; correctness verification is an issue, and so is scalability [4]. In contrast, our method does not require specific software since RESTdesc has been designed to work with generic reasoners.

3 Web API composition

Two partial problems are involved in creating a composition:

Matching is the decision whether the invocation of a Web API call A_n is sufficient (and somehow necessary) to execute another call A_m. In logical terms, this means:
$$A_m(c_i, c_j) \Rightarrow A_n(c_j, c_k) \qquad \text{with } \neg A_n(c_i, c_k)$$
So the postconditions c_j of calling A_m with preconditions c_i enable calling A_n, whereas the preconditions c_i alone are not sufficient to call A_n.

Chaining is the repeated application of matching, to generate a chain of Web API calls $A_1 \dots A_n$ where every two successive calls match, *i.e.*, $\forall m < n$ *match*(A_m, A_{m+1}).

However, in the most general sense, a composition is not a chain of calls, but rather a graph in which multiple calls can provide the preconditions for another call. This indicates that the problem of composition is more complicated than simply determining all matches and identifying chains.

Instead, we define a composition as an acyclic graph of Web API calls and distinguish between two kinds of compositions. In *exploration-based composition*, the task is to identify graphs of API calls that are possible given the current application state. In *goal-driven* compositions, an external agent (such as the user) indicates a certain goal that needs to be accomplished, and the composition is then a plan that satisfies this goal's conditions, starting from the current state.

4 Proof-based composition

The composition method we present in this paper is based on the close relationship of logic implication and composition, and the fact that RESTdesc descriptions are in fact logic implication rules. The generalized notion of compositions as a graph can be expressed as an implication:

$$A_n(c_n, c'_n) \wedge \ldots \wedge A_m(c_m, c'_m) \Rightarrow A_k(c_k, c'_k)$$

or indeed, as logical entailment:

$$A_n(c_n, c'_n), \ldots, A_m(c_m, c'_m) \vdash A_k(c_k, c'_k)$$

In the above case, the Web API calls A_n to A_m are necessary to execute A_k. Stated differently, A_k is *provable* with A_n to A_m. This means that the generation of a composition essentially comes down to generating a proof that the invocation of selected Web APIs leads to the fulfilment of a predefined goal, wherein a goal is defined as a desired end state entailed by the combined postconditions. Concretely, to determine whether a goal g can be reached from an initial state i, we should construct a series of implications formed by Web API calls, which will form an implication graph that proves g from i.

RESTdesc describes Web APIs indeed as logic rules, expressed in Notation3, which has an associated logic framework called N3Logic [1]. Therefore, composing RESTdesc-described APIs is possible by building an N3Logic proof with these descriptions. Several generic Notation3 reasoners exist and due to the design of RESTdesc descriptions, they all are natively capable of generating RESTdesc compositions.

The reasoner is supplied with the following inputs:

The initial state as an RDF document, which describes currently available resources and their state.

Web API descriptions in RESTdesc format of various APIs the user has access to.

The goal state *(optional)* as an RDF document, which describes the desired resource state.

If the goal state is omitted, the composition will be exploration-based, meaning the reasoner will construct compositions where the initial state is a precondition (possibly limited to a certain number of compositions or to compositions of a certain length). If the goal state is available, the composition will be goal-driven, meaning that only compositions will be found that achieve the specified goal.

5 Evaluation

An important question is whether the proposed solution performs well on a Web scale, thus with hundreds of Web APIs and for compositions of various lengths. We have conducted experiments with the EYE and cwm reasoners using a benchmark suite [5] that is freely available online.[3] The benchmark consists of a description generator, which purposely creates descriptions that match on a given initial and goal state, and a benchmarking utility that evaluates the performance.

The results indicated that proof-based composition is possible even for complex compositions (*e.g.*, with more than 1,000 API calls). The fastest reasoner in the test, EYE, was able to create most compositions in well under one second on an average consumer computer.

[3] http://github.com/RubenVerborgh/RESTdesc-Composition-Benchmark

6 Conclusion and future work

In this paper, we have presented a reasoner-based composition algorithm for RESTdesc-described Web APIs. A performance evaluation has indicated that our method is sufficiently fast to perform on a Web scale.

We currently have an implementation in the domain of sensor networks [5], but plan to test this approach to other domains. One interesting direction we plan to investigate is multimedia analysis and adaptation, thus communicating with multimedia algorithms through Web APIs and then chaining them together to perform complex actions in an automated way, where compositions are dynamically generated according to the user's needs.

Acknowledgements

The described research activities were funded by Ghent University, the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union.

References

1. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008), http://arxiv.org/abs/0711.1533
2. DuVander, A.: 8,000 APIs: Rise of the enterprise (Nov 2012), http://blog.programmableweb.com/2012/11/26/8000-apis-rise-of-the-enterprise/
3. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. Transactions on Internet Technology 2(2), 115–150 (May 2002), http://dl.acm.org/citation.cfm?id=514185
4. Milanovic, N., Malek, M.: Current solutions for Web service composition. IEEE Internet Computing 8(6), 51–59 (Nov 2004)
5. Verborgh, R., Haerinck, V., Steiner, T., Van Deursen, D., Van Hoecke, S., De Roo, J., Van de Walle, R., Gabarró Vallés, J.: Functional composition of sensor Web APIs. In: Proceedings of the 5th International Workshop on Semantic Sensor Networks (Nov 2012), http://ceur-ws.org/Vol-904/paper6.pdf
6. Verborgh, R., Steiner, T., Van Deursen, D., Coppens, S., Gabarró Vallés, J., Van de Walle, R.: Functional descriptions as the bridge between hypermedia APIs and the Semantic Web. In: Proceedings of the Third International Workshop on RESTful Design. pp. 33–40. ACM (Apr 2012), http://www.ws-rest.org/2012/proc/a5-9-verborgh.pdf
7. Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Van de Walle, R., Gabarró Vallés, J.: Description and Interaction of RESTful Services for Automatic Discovery and Execution. In: Proceedings of the FTRA 2011 International Workshop on Advanced Future Multimedia Services (Dec 2011)
8. Verborgh, R., Steiner, T., Van Deursen, D., De Roo, J., Van de Walle, R., Gabarró Vallés, J.: Capturing the functionality of Web services with functional descriptions. Multimedia Tools and Applications (2013), http://www.springerlink.com/index/d041t268487gx850.pdf