HeadREST: A Specification Language for RESTful APIs

Vasco T. Vasconcelos[1], Antónia Lopes[1], and Francisco Martins[2]

LASIGE, University of Lisbon,[1] University of Azores,[2] Portugal

This work is about an industrial application of type theory technology. We have developed HeadREST, a language to model REST services that goes well beyond the descriptive power of current languages. HeadREST is at the basis of an ambitious project addressing fundamental aspects of the API lifecycle; the language aims at supporting testing, validation, runtime monitoring, and code generation.

REST (Representational State Transfer) is an architectural style regarded as an abstract model of the web architecture and based on the concept of resource [3]. According to Fielding, a resource R is a function $M_R(t)$ that associates to each time instant t a set of values, which can be *identifiers* or *representations of resources* [4]. Identifiers are used to distinguish the resource involved in an interaction. Representations capture the current state or the intended state of the resource, and are used to perform actions on the resource.

We focus on REST applications that communicate over HTTP and interact with external systems through web resources identified by Unique Resource Identifiers (URIs). Thus, actions that can be performed on a resource correspond to requests for the execution of the methods offered by HTTP (GET, POST, PUT, and DELETE). The metadata and data to be transferred are sent, respectively, in the header and in the body of the request. As a result to a request a response is produced containing metadata and data to be transferred back to the customer.

Different *interface description languages* (IDLs) have been purposely designed to support the formal description of REST APIs. The most representative are probably Open API Initiative [6] (originally called Swagger), the RESTful API Modeling Language [7] (RAML), and API Blueprint [1]. These IDLs allow a detailed description of the syntactic aspects of the data transferred in REST interactions and are associated to a large number of tools. Being focused on the structure of the data exchanged, they ignore important semantic aspects, including relating different input/output data, the input against the state of the service, and the output against the input.

Our approach is based on two key ideas:

- *Types* to express properties of states and of data exchanged in interactions and
- *Pre- and post-condition* to express the relationship between data sent in requests and that obtained in responses, as well as the resulting state changes.

These ideas are embodied in HeadREST, a language built on the two fundamental concepts of DMinor [2]:

- *Refinement types*, `x:T where e`, consisting of values `x` of type `T` that satisfy property `e`, and
- A *predicate*, `e in T`, which returns true or false depending on whether the value of expression `e` is or is not of type `T`.

HeadREST allows to formally describe properties of data and to observe state changes of REST APIs through a collection of assertions. Assertions take the form of Hoare triples [5]. In $\{\phi\}$ $(a\ t)$ $\{\psi\}$, a is an action (`GET`, `POST`, `PUT`, or `DELETE`), t is an *URI template*, and ϕ and ψ are boolean expressions. Formula ϕ, called the *precondition*, addresses the state in

which the action is performed as well as the data transmitted in the request, whereas ψ, the *postcondition*, addresses the state resulting from the execution of the action together with the values transmitted in the response. The assertion reads

> If a request for the execution of a over an expansion of t carries data satisfying ϕ and the action is performed in a state satisfying ϕ, then the data transmitted in the response satisfies ψ and so does the state resulting from the execution of the action.

A simple contact management system could be based on a *new type*

```
resource Contact
```

There may be many *representations* of such a resource. Here is one:

```
type NameAndEmail = {
  name: (x: string where matches(/^[a-zA-Z]{3,15}$/, x)),
  email: (x: string where contains("@", x))
}
```

An *assertion* describing a successful contact creation could be written as

```
{request in {body: NameAndEmail} &&
 ∀c:Contact. ∀r:NameAndEmail. r repof c ⇒ request.body.name ≠ r.name
}
POST /contacts
{response.code == 200 &&
 response in {body: NameAndEmail, header: {Location: URI}} &&
 ∃c:Contact. response.body repof c && response.header.Location uriof c
}
```

where `request` and `response` are builtin identifiers, and predicates `repof` and `uriof` describe values associated to resources. The precondition asks the new contact name to be unique across all contacts and their representations. In such a case, the postcondition signals success (`code` 200) and states that `response` includes a representation and an URI of the newly created `Contact` resource.

We have used HeadREST to describe different APIs, including a part of GitLab (800 lines of spec code). We have developed an Eclipse plugin to validate HeadREST specifications and a tool to automatically test REST APIs against specifications. These tools rely on an external SMT to solve the semantic subtyping goals required by D-Minor [2]. We are further working on a tool to generate server stubs and client SDKs from HeadREST specifications.

References

[1] API blueprint. https://apiblueprint.org/, 2018. Accessed 25-Jan-2018.

[2] Gavin M. Bierman, Andrew D. Gordon, Catalin Hritcu, and David E. Langworthy. Semantic subtyping with an SMT solver. *J. Funct. Program.*, 22(1):31–105, 2012.

[3] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[4] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Techn.*, 2(2):115–150, 2002.

[5] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[6] Open API Initiative. https://www.openapis.org, 2018. Accessed 26-Jan-2018.

[7] RESTful API modeling language. https://raml.org, 2018. Accessed 25-Jan-2018.