LASIGE reliable software systems

# Statically Checking REST API Consumers

Nuno Burnay, Antónia Lopes and Vasco T. Vasconcelos

LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

ULISBOA Ciências ULisboa FCT Fundação para a Ciência e a Tecnologia

# REST APIs



- 69% of the internet APIs are handed by REST services
- RESTful services are considered more flexible and offering more control
- Large software companies like Facebook, Google and Microsoft have REST APIs endpoints to their more relevant services

# REST Calls 101

LASIGE reliable software systems

- Request
  1. HTTP method
  2. Location (URL)
  3. Headers
  4. Body

```
POST /api/v1/create HTTP/1.1
Host: dummy.restapiexample.com
Content-Type: application/json

{"name": "Ada", "salary": "123",
"age": "23"}
```

- Response
  1. Code
  2. Headers
  3. Body

```
HTTP/1.1 200 OK
Content-Type: application/json

{"name": "Ada", "salary": "123",
"age": "23", "id": "5407"}
```

# Real example



```
1 // Get Location ID from lat/long coordinates
2 function getLocation() {
3     var client_id = '813efa314de94e618290bc8bfcbbb1ac';
4     // URL for API request to find locations
5     var locationUrl =
6         'https://api.instagram.com/v1/locations/search?lat='+LAT
7             +'&lng='+LNG+'&distance=5000&client_id='+client_id;
8     var result = $.ajax({
9         url: locationUrl,
10        dataType: "jsonp",
11        type: "GET",
12    })
13    .done(function(result){
14        //grab first ID from results
15        var locationId = result.data[0].id;
16        return locationId
17    })
18 }
```

URL query parameters

Operation

Response Body

# The problem

LASIGE reliable software systems

"Clients of web APIs […] do not know whether the signature of a call (i.e., the URL, the request payload, query parameters) is valid until run-time"

Wittern et al. (2017)
Opportunities in Software Engineering Research for Web API Consumption

"[…] most faults can be attributed to the invalid or missing request data, and most API consumers seem to be impacted by faults caused by invalid request data"

Aué et al. (2018)
An Exploratory Study on Faults in Web API Integration in a Large-Scale Payment Company

Misuse of responses in consumer code can only be found at run-time (even when models of response data exist)

# Real example

LASIGE reliable software systems

```
1 // Get Location ID from lat/long coordinates
2 function getLocation() {
3     var client_id = '813efa314de94e618290bc8bfcbbb1ac';
4     // URL for API request to find locations
5     var locationUrl =
6         'https://api.instagram.com/v1/locations/search?lat='+LAT
7             +'&lng='+LNG+'&distance=5000&client_id='+client_id;
8     var result = $.ajax({
9         url: locationUrl,
10        dataType: "jsonp",
11        type: "GET",
12    })
13    .done(function(result){
14        //grab first ID from results
15        var locationId = result.data[0].id;
16        return locationId
17    })
18 }
```

Valid URL and query parameters?

Has the response body the expected format?

1. Is `result` an object? If yes,
2. Is `data` a field in the object? If yes,
3. Is `result.data` a non empty array?
4. Is `id` a field of the first element?

# Our aproach

LASIGE reliable software systems

- **HeadREST**, a specification language for REST APIs
  - Captures important properties of APIs that cannot be expressed in currently available IDLs such as OpenAPI
- **SafeRestScript**, a language for programming client code
  - Direct support for REST calls
  - Expressive type

# HeadREST

An expressive specification language for describing REST APIs

LASIGE reliable software systems

U LISBOA | Ciências ULisboa | FCT Fundação para a Ciência e a Tecnologia

# HeadREST

LASIGE reliable software systems

- 2 key ideas:
  - Types to express properties of states and of data exchanged in interactions;
  - Pre and post-conditions to express the relationship between data sent in requests and those obtained in responses, as well as the resulting state changes.
- Pre and post-conditions relations are expressed using Hoare Triples:

$$\{e_1\}\; m\; u\; \{e_2\}$$

# Type system



- The type system is based on the Dminor language (Bierman et al., 2009)
- Expressive types thanks to two primitives:
  - Refinement types

    ```
    (x:T where e)
    ```

    consisting of values `x` of type `T` that satisfy property `e`
  - 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`

# Syntax

LASIGE reliable software systems

$$
\begin{array}{lrcl}
\text{Scalar types} & G & ::= & \mathsf{Integer} \mid \mathsf{String} \mid \mathsf{Boolean} \mid \{\} \mid \mathsf{Regexp} \mid \mathsf{URITemplate} \\
\text{Types} & T & ::= & \mathsf{Any} \mid G \mid \{l\colon T\} \mid T[] \mid (x\colon T\ \mathsf{where}\ e) \\
\text{Constants} & c & ::= & n \mid s \mid \mathsf{true} \mid \mathsf{false} \mid \{\} \mid u \mid r \mid \mathsf{null} \\
\text{Expressions} & e & ::= & x \mid c \mid f(e_1, \ldots, e_n) \mid e\,?\,e : e \mid e\ \mathsf{in}\ T \mid \{l_1 = e_1, \ldots, l_n = e_n\} \\
 & & & \mid e.l \mid [e_1, \ldots, e_n] \mid e[e] \mid \mathsf{forall}\ x\colon T.e \mid \mathsf{exists}\ x\colon T.e \\
\text{Verbs} & m & ::= & \mathsf{get} \mid \mathsf{put} \mid \mathsf{post} \mid \mathsf{delete} \\
\text{Declarations} & D & ::= & \{e\}m\,u\{e\}; D \mid \epsilon
\end{array}
$$

Examples of derived types:

$T \mathbin{\&} U \triangleq x\colon \mathsf{Any\ where}\ (x\ \mathsf{in}\ T \mathbin{\&} x\ \mathsf{in}\ U)$

$!T \triangleq x\colon \mathsf{Any\ where}\ !(x\ \mathsf{in}\ T)$

$\{?l\colon T\} \triangleq x\colon \{\}\ \mathsf{where}\ x\ \mathsf{in}\ \{l\colon \mathsf{Any}\} \Rightarrow x\ \mathsf{in}\ \{l\colon T\}$

$\mathsf{Natural} \triangleq x\colon \mathsf{Integer\ where}\ x \geq 0$

# Example

LASIGE reliable software systems

```
1  specification Instagram
2
3  type SearchLocation = (o: {
4        ?distance: (x: Integer where x>0 && x<=5000),
5        ?lat: Integer,
6        ?lng: Integer,
7        ?facebook_places_id: String,
8        ?foursquare_id: String
9  } where
10       isdefined(lat) <=> isdefined(lng) &&
11       (isdefined(lat) || isdefined(facebook_places_id) || isdefined(foursquare_id))
12 )
13
14 type Location = {id: String, name: String, lat: Integer, lng: Integer}
15
16 { !(request in {template: SearchLocation}) }
17       get `/locations/search{?distance,lat,lng,facebook_places_id,foursquare_id,client_id}`
18 { response.code != 200 }
19
20 { true }
21       get `/locations/search{?distance,lat,lng,facebook_places_id,foursquare_id,client_id}`
22 { response.code == 200 ==> response in {body: {data: Location[]}} }
```

Refining input (line 11)

Unsuccess case (lines 16–18)

Sucess case (lines 20–22)

LASIGE reliable software systems

# SafeRESTScript

A subset of JavaScript with an expressive type system supporting the validation of REST calls

U LISBOA
Ciências ULisboa
FCT Fundação para a Ciência e a Tecnologia

# Key ideas

LASIGE reliable software systems

- A type-safe variant of JavaScript
- Supports primitively REST calls
- Transpiles to JavaScript
- Incorporates HeadREST expressive type system
- Flow typing: the type of an (imperative) variable changes with the control flow
  - **Declared type**: stays constant along the program
  - **Effective type**: the set of values that a variable may have at a given point in a program (a subtype of the declared type)

# Syntax

LASIGE reliable software systems

| | | | |
|---|---|---|---|
| Constants | $c$ | $::=$ | $\ldots \mid \mathsf{undefined}$ |
| Expressions | $e$ | $::=$ | $\ldots \mid \mathsf{await}^{?}\; m\, u\, e$ |
| Locations | $w$ | $::=$ | $x \mid w.l \mid w[e]$ |
| Statements | $S$ | $::=$ | $w = e \mid \mathsf{if}\,(e)\,\mathsf{then}\; S\; \mathsf{else}\; S \mid \mathsf{while}\,(e)\,\mathsf{inv}\; e\; S \mid \mathsf{return}\; e \mid S; S \mid \epsilon$ |
| Declarations | $D$ | $::=$ | $\mathsf{specification}\; s\; \mathsf{of}\; u \mid \mathsf{var}\; T\, x = e \mid \mathsf{async}^{?}\; T\, x\, (\overline{T\, x})\, \{\overline{T\, x = e}; S\}$ |
| Programs | $P$ | $::=$ | $D; P \mid \epsilon$ |

# A static checker

LASIGE reliable software systems

- Thanks to its expressive type system, SafeRestScript can statically detect common runtime errors, such as:
  - Division by zero
  - Array access outside bounds
  - Null dereference
  - …

```
int f(nat x) {
    return 42 / x; // Error!
}
```

Would require $x$ of type $(y\colon \text{int where } y \neq 0)$

```
void g(int[] a) {
    a[0] = 1; // Error!
}
```

Would require $a$ of type $(y\colon \text{int[] where length}(a) > 0)$

# Validation

LASIGE reliable software systems

- Refinement types and flow typing demand a semantic validation of candidate programs
- Validation relies on the ability to synthesize the effective type of a general expression in a given context
- We accomplish typing validation by translating SRS programs to Boogie intermediate language

# Translation to Boogie

LASIGE reliable software systems

- Builds a set of type-related assertions, and uses the Boogie tool to check whether the assertions hold
- Types are not translated to Boogie, but the *in type* relation is
- Boogie generates verification conditions, which it offloads to an SMT solver
- The axiomatization of the typing relation is inspired by Whiley (Utting et al., 2017)

# Example



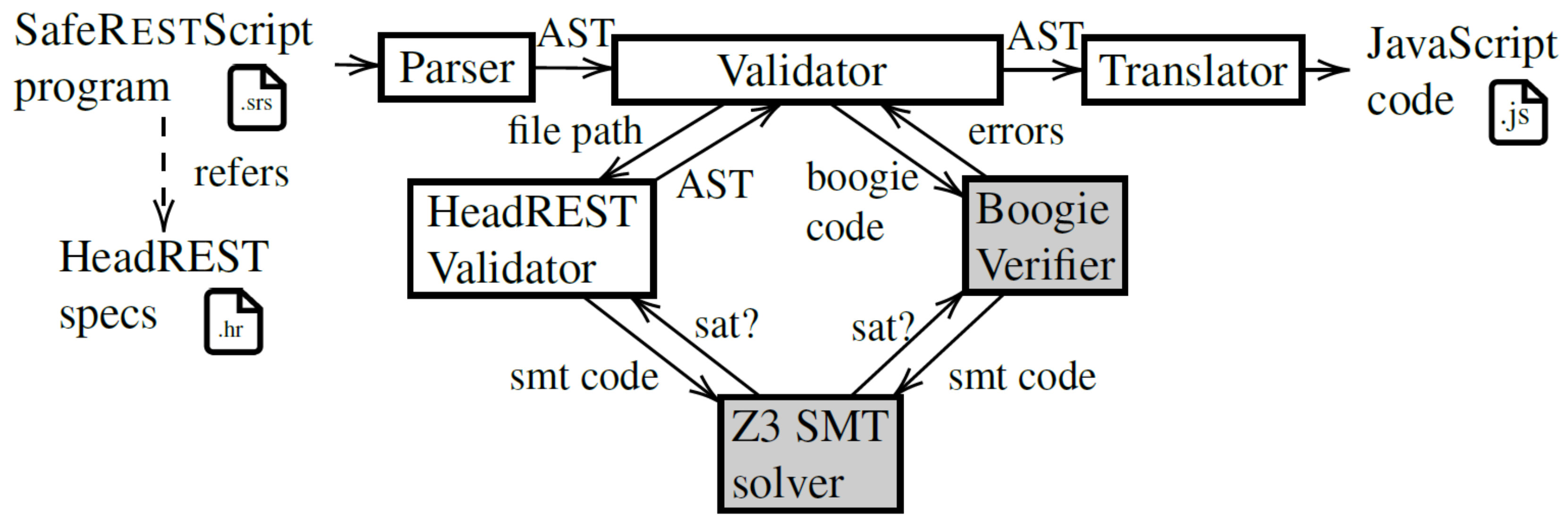
```
1  specification "./specs/Instagram.hrest" of "https://api.instagram.com/v1"
2
3  var string client_id = "813efa314de94e618290bc8bfcbbb1ac";
4
5  type Error = {error: string};
6
7  async string|Error getLocation(int lat, int lng) {
8      SearchLocation searchLocation = {lat= lat, lng= lng, distance= 5000, client_id= client_id};
9      Request request = {template = searchLocation};
10     Response result = await get
11             "/locations/search{?distance,lat,lng,facebook_places_id,foursquare_id,client_id}"
12              request;
13     if (result.code != 200) {
14             return {error = "No locations found!"};
15     }
16     return result.body.data[0].id;
17 }
```

Index out of bounds

According to `Instagram.hrest` specification, the array can be empty

# Compilation

LASIGE reliable software systems






U LISBOA Ciências ULisboa FCT Fundação para a Ciência e a Tecnologia

# Evaluation

A series of tests that consume well known APIs

LASIGE reliable software systems

U LISBOA Ciências ULisboa FCT Fundação para a Ciência e a Tecnologia

# Evaluation

LASIGE reliable software systems

- Goal: evaluate to what degree can SRS be used in examples which include complex REST calls that can be found in real examples
- We used SafeRestScript to write programs that consume publicly available APIs:
  - PetStore
  - DummyAPI
  - Instagram
  - GitHub
  - GitLab
- We also wrote HeadREST specifications of part of these APIs

# Performance results

LASIGE reliable software systems

| | HeadREST | | | | SRS | | | |
|---|---|---|---|---|---|---|---|---|
| | #EndP | #Types | LOC | Check (s) | #EndP | #Func | LOC | Check (s) |
| Instagram#app1 | 6 | 9 | 225 | 1.3 | 3 | 4 | 82 | 1.5 |
| Instagram#app2 | 6 | 9 | 225 | 1.3 | 3 | 3 | 65 | 1.8 |
| GitHub | 5 | 9 | 93 | 0.8 | 5 | 3 | 86 | 1.3 |
| GitLab | 10 | 20 | 435 | 1.7 | 8 | 10 | 250 | 50.5 |

The complexity of the types involved in REST calls significantly slows the validation process when the correctness of the code strongly depends on types

# Conclusions of the evaluation

LASIGE reliable software systems

- HeadREST supports the specification of a variety of API endpoints found in real examples
- HeadREST is able to capture important properties of these endpoints that were previously available only in natural language
- The formalisation of properties allowed SafeRestScript to find all sort of errors in our code:
  - invalid or missing data in the requests
  - errors in the use of the data received in the response
- Most of these errors would not be found in JavaScript code (Wittern et al., 2017)

# In a nutshell…

LASIGE reliable software systems

- HeadREST is an expressive specification language for describing REST APIs
- SafeRestScript is a client-side language that statically validates REST calls against HeadREST specifications
- Both these languages contribute to the verification and correction in REST APIs consumer code

# Future work



- Introduce references: the lack of references is the most relevant difference between SRS and JavaScript
  - A preliminary experience showed that this change substantially increases the validation time
- Take advantage of JavaScript standard libraries
  - Possibly by following the TypeScript approach by introducing declaration files where external JavaScript can be declared
- Detect inconsistent triples in HeadREST specifications
  - Specifications featuring inconsistent triples allow programs with typing errors to be incorrectly validated

LASIGE reliable software systems

# SafeRESTScript

ULISBOA Ciências ULisboa FCT Fundação para a Ciência e a Tecnologia