Visual Modeling of RESTful Conversations with RESTalk

Ana Ivanchikj, Cesare Pautasso
Università della Svizzera italiana (USI)
Lugano, Switzerland

Silvia Schreier
innoQ Deutschland GmbH
Monheim, Germany
MOTIVATION

What is a RESTful conversation?

Why do we need a Domain Specific Language to model it?
RESTful conversation

Motivation ➔ RESTalk ➔ RESTalk Exploratory Survey ➔ Conclusions

GET URI ➔ PUT URI ➔ GET URI

303 See Other ➔ 200 OK

Link Relationships (Hypermedia)

RESTful Conversations ➔ REST API Structure ➔ REST API dynamics
RESTful conversation

![Diagram of RESTful conversation with URIs and relationships](image)

- GET URI
- PUT URI
- 200 OK
- 201 Created
- 303 See Other
- Link Relationships (Hypermedia)
- reference
- create

**Motivation** ➔ **RESTalk** ➔ **RESTalk Exploratory Survey** ➔ **Conclusions**
REST API structure - RAML

```
* api.raml
1
2
3
title: World Music API
4
5
version: v1
6
7
baseUri:
8
9
schemas:
10
11
resourceTypes:
12
13
traits:
14
15
/songs:
16
17
type: { collection: { schema: song } }
18
19
is: { secured }
20
21
/{songId}:
22
23
type: { member: { schema: song } }
24
25
is: { secured }
```

RAML editor

World Music API

/songs

Type: collection

Description

No description.

Parameters

Response

200
OK

JSON SCHEMA

```json
{
"type": "object",
"description": "The canonical song representation",
"properties": {
  "title": { "type": "string" },
  "artist": { "type": "string" }
}
"required": [ "title", "artist" ]
}
```
REST API structure - Swagger

POST /pet

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;id&quot;: 0,</td>
</tr>
<tr>
<td></td>
<td>&quot;category&quot;: {</td>
</tr>
<tr>
<td></td>
<td>&quot;id&quot;: 0,</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot;: &quot;string&quot;</td>
</tr>
<tr>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td>&quot;name&quot;: &quot;doggie&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;photoUrls&quot;: [</td>
</tr>
<tr>
<td></td>
<td>]</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Parameter content type: application/json

Description

Pet object that needs to be added to the store

Parameter Type | Data Type
---|---
body | body

Model | Model Schema
---|---

Click to set as parameter value

Response Messages

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Reason</th>
<th>Response Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>405</td>
<td>Invalid input</td>
<td></td>
</tr>
</tbody>
</table>

Try it out!
REST API dynamics

- Redirect -

GET /resource1 HTTP/1.1

HTTP/1.1 303 See Other
Location: /resource2

GET /resource2 HTTP/1.1

HTTP/1.1 200 OK

UML Sequence diagram

BPMN Choreography diagram

RESTalk

GET /resource1

303 See Other
Location: /resource2

GET /resource2

200 OK
RESTalk constructs

Long-running request modeled with RESTalk
RESTalk

Events
- Start event
- End event
- Timer event

Flows
- Sequence flow
- Hyperlink flow

Gateways
- Parallel AND gateway
- Exclusive XOR gateway
- Inclusive OR gateway
Long-running Request
-RESTful Conversation Pattern Example-
-Visualized with RESTalk-

e.g.: http://docs.aws.amazon.com/amazonglacier/latest/dev/job-operations.html
## Long-running Request

- **Happy path** -

<table>
<thead>
<tr>
<th>Action</th>
<th>Method</th>
<th>URL</th>
<th>Status Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create job</td>
<td>POST</td>
<td>/job</td>
<td>HTTP/1.1</td>
<td>202</td>
</tr>
<tr>
<td>Poll</td>
<td>GET</td>
<td>/job/42</td>
<td>HTTP/1.1</td>
<td>200</td>
</tr>
<tr>
<td>Read results</td>
<td>GET</td>
<td>/job/42/output</td>
<td>HTTP/1.1</td>
<td>200</td>
</tr>
</tbody>
</table>

### Diagram

- POST /job
- 202 Accepted Location: /job/42
- GET /job/42
- 200 OK
- GET /job/42/output
- 200 OK
- 303 See Other Location: /job/42/output
- GET /job/42/output
- 200 OK
Long-running Request
- Resending the request -

<table>
<thead>
<tr>
<th>Action</th>
<th>Method</th>
<th>Endpoint</th>
<th>Status</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create job</td>
<td>POST</td>
<td>/job</td>
<td>HTTP/1.1</td>
<td></td>
</tr>
<tr>
<td>Poll</td>
<td>GET</td>
<td>/job/42</td>
<td>HTTP/1.1</td>
<td>/job/42</td>
</tr>
<tr>
<td>Read results</td>
<td>GET</td>
<td>/job/42/output</td>
<td>HTTP/1.1</td>
<td>/job/42/output</td>
</tr>
</tbody>
</table>

POST /job
202 Accepted
Location: /job/42

GET /job/42
200 OK

GET /job/42/output
303 See Other
Location: /job/42/output

GET /job/42/output
200 OK
### Long-running Request

- **Reading the results** -

<table>
<thead>
<tr>
<th>Method</th>
<th>Request URI</th>
<th>HTTP Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/job</td>
<td>HTTP/1.1</td>
</tr>
<tr>
<td>POST</td>
<td>/job</td>
<td>HTTP/1.1</td>
</tr>
</tbody>
</table>

**POST /job**

- 202 Accepted
- Location: /job/42

**Poll**

<table>
<thead>
<tr>
<th>Method</th>
<th>Request URI</th>
<th>HTTP Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/job/42</td>
<td>HTTP/1.1</td>
</tr>
<tr>
<td>GET</td>
<td>/job/42</td>
<td>HTTP/1.1</td>
</tr>
</tbody>
</table>

**GET /job/42**

- 303 See Other
- Location: /job/42/output

**Read results**

<table>
<thead>
<tr>
<th>Method</th>
<th>Request URI</th>
<th>HTTP Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/job/42/output</td>
<td>HTTP/1.1</td>
</tr>
<tr>
<td>GET</td>
<td>/job/42/output</td>
<td>HTTP/1.1</td>
</tr>
</tbody>
</table>

**GET /job/42/output**

- 200 OK
Long-running Request
- Deleting the output resource -

Create job

POST /job HTTP/1.1

POST /job HTTP/1.1

HTTP/1.1 202 Accepted
Location: /job/42

Poll

GET /job/42 HTTP/1.1

HTTP/1.1 200 OK

GET /job/42 HTTP/1.1

HTTP/1.1 303 See Other
Location: /job/42/output

Read results

GET /job/42/output HTTP/1.1

HTTP/1.1 200 OK

GET /job/42/output HTTP/1.1

HTTP/1.1 200 OK

Delete output

DELETE /job/42/output HTTP/1.1

HTTP/1.1 200 OK

DELETE /job/42/output

200 OK
Long-running Request
- Deleting the output resource -

<table>
<thead>
<tr>
<th>Create job</th>
<th>POST /job HTTP/1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POST /job HTTP/1.1</td>
</tr>
<tr>
<td></td>
<td>HTTP/1.1 202 Accepted</td>
</tr>
<tr>
<td></td>
<td>Location: /job/42</td>
</tr>
<tr>
<td>Poll</td>
<td>GET /job/42 HTTP/1.1</td>
</tr>
<tr>
<td></td>
<td>HTTP/1.1 200 OK</td>
</tr>
<tr>
<td></td>
<td>GET /job/42 HTTP/1.1</td>
</tr>
<tr>
<td></td>
<td>HTTP/1.1 303 See Other</td>
</tr>
<tr>
<td></td>
<td>Location: /job/42/output</td>
</tr>
<tr>
<td>Delete output</td>
<td>DELETE /job/42/output HTTP/1.1</td>
</tr>
<tr>
<td></td>
<td>HTTP/1.1 200 OK</td>
</tr>
</tbody>
</table>
Long-running Request
- Deleting the job resource -

<table>
<thead>
<tr>
<th>Action</th>
<th>Method</th>
<th>URL</th>
<th>Status Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create job</td>
<td>POST</td>
<td>/job</td>
<td>HTTP/1.1</td>
<td>202 Accepted</td>
</tr>
<tr>
<td>Poll</td>
<td>GET</td>
<td>/job/42</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
</tr>
<tr>
<td>Delete output</td>
<td>DELETE</td>
<td>/job/42/output</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
</tr>
<tr>
<td>Clean up</td>
<td>DELETE</td>
<td>/job/42</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
</tr>
</tbody>
</table>

POST /job
HTTP/1.1 202 Accepted
Location: /job/42

GET /job/42
HTTP/1.1 200 OK

GET /job/42/output
HTTP/1.1 200 OK

DELETE /job/42
HTTP/1.1 200 OK

DELETE /job/42/output
HTTP/1.1 200 OK
Long-running Request
- Deleting the job resource -

<table>
<thead>
<tr>
<th>Operation</th>
<th>Method</th>
<th>Path</th>
<th>Status Code</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create job</td>
<td>POST</td>
<td>/job</td>
<td>HTTP/1.1</td>
<td>POST /job HTTP/1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HTTP/1.1 202 Accepted</td>
</tr>
<tr>
<td>Delete job</td>
<td>DELETE</td>
<td>/job/42</td>
<td>HTTP/1.1</td>
<td>DELETE /job/42 HTTP/1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HTTP/1.1 200 OK</td>
</tr>
</tbody>
</table>

POST /job

202 Accepted
Location: /job/42

GET /job/42

200 OK

303 See Other
Location: /job/42/output

GET /job/42/output

200 OK

DELETE /job/42

200 OK

DELETE /job/42/output

200 OK
Long-running Request

POST /job

202 Accepted
Location: /job/42

DELETE /job/42/output

200 OK

DELETE /job/42

200 OK

GET /job/42

200 OK

303 See Other
Location: /job/42/output

GET /job/42/output

200 OK

DELETE /job/42/output

200 OK

Mo6va6onèRESTalkèRESTalk

Exploratory	
  Conclusions
### Long-running Request

**- Short happy path -**

<table>
<thead>
<tr>
<th>Create job</th>
<th>POST /job HTTP/1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP/1.1 202 Accepted Location: /job/42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Poll</th>
<th>GET /job/42 HTTP/1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP/1.1 303 See Other Location: /job/42/output</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Read results</th>
<th>GET /job/42/output HTTP/1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP/1.1 200 OK</td>
</tr>
</tbody>
</table>

**POST /job**

**HTTP/1.1**

**202 Accepted**

**Location:** /job/42

**GET /job/42**

**HTTP/1.1**

**200 OK**

**Location:** /job/42/output

**POST /job/42/output**

**HTTP/1.1**

**200 OK**

**DELETE /job/42/output**

**HTTP/1.1**

**200 OK**

**DELETE /job/42**

**HTTP/1.1**

**200 OK**

**GET /job/42/output**

**HTTP/1.1**

**200 OK**

**DELETE /job/42/output**

**HTTP/1.1**

**200 OK**

**DELETE /job/42**

**HTTP/1.1**

**200 OK**

**GET /job/42**

**HTTP/1.1**

**303 See Other**

**Location:** /job/42/output

**GET /job/42/output**

**HTTP/1.1**

**200 OK**

**DELETE /job/42/output**

**HTTP/1.1**

**200 OK**

**GET /job/42**

**HTTP/1.1**

**200 OK**

**DELETE /job/42**

**HTTP/1.1**

**200 OK**
Long-running Request

POST /job
202 Accepted
Location: /job/42

DELETE /job/42/output
200 OK

DELETE /job/42
200 OK

GET /job/42
200 OK
303 See Other
Location: /job/42/output

GET /job/42/output
200 OK

DELETE /job/42/output
200 OK
## Long-running Request

**Long path**

<table>
<thead>
<tr>
<th>Action</th>
<th>Request Method</th>
<th>URI</th>
<th>HTTP Status</th>
<th>Response Message</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create job</td>
<td>POST</td>
<td>/job HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>202 Accepted</td>
<td>Location: /job/42</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>/job HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td>Poll</td>
<td>GET</td>
<td>/job/42 HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET</td>
<td>/job/42 HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET</td>
<td>/job/42 HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET</td>
<td>/job/42 HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>303 See Other</td>
<td>Location: /job/42/output</td>
</tr>
<tr>
<td>Read results</td>
<td>GET</td>
<td>/job/42/output HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GET</td>
<td>/job/42/output HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td>Delete output</td>
<td>DELETE</td>
<td>/job/42/output HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
<tr>
<td>Delete job</td>
<td>DELETE</td>
<td>/job/42 HTTP/1.1</td>
<td>HTTP/1.1</td>
<td>200 OK</td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

- POST /job
  - 202 Accepted
  - Location: /job/42
- GET /job/42
  - 200 OK
- GET /job/42/output
  - 200 OK
- DELETE /job/42/output
  - 200 OK
- DELETE /job/42
  - 200 OK
RESTalk Exploratory Survey

Motivation
Goals
Design
Results
Approach: Agile DSML Design

Survey Experiment

Feedback implementation

RESTalk Use

Final Goal:
*Increase acceptance and dissemination of RESTalk*
Explorative Survey Goals

• **Goal 1:** Evaluate the need in industry for a domain specific language for modeling RESTful conversations

• **Goal 2:** Evaluate the cognitive characteristics of RESTalk

*Qualitative research technique => No statistical inference*
Exploratory Survey Design

English + German

Modeling RESTful Conversations (English version)

Welcome to the RESTful conversations survey. RESTful conversations are complex interactions between client(s) and server(s). For more details on the conversation based approach for modeling RESTful APIs please refer to the following paper: http://design.inf.usi.ch/sites/default/files/biblio/wicsa2015.pdf.

*Please note that having in mind the exploratory goal of this survey, going back to the previous question is not an option in the same.

Usage of BPMN Choreography

Why did you decide to use BPMN Choreography?

Which constructs of BPMN Choreography do you appreciate the most and you find core for depicting RESTful conversations?

7 question groups:
- demographic data
- background on used notations in practice
- RESTalk’s intuitiveness
- RESTalk vs. standard BPMN Choreography
- reading task
- modeling task
- RESTalk’s evaluation

Mainly open ended optional questions
Respondents’ Demographic Data

35 respondents:
- 74% industry
- 26% academia

Profile:
- IT consultants
- SW quality engineers
- SW developers
- SW architects
- a CTO
- researchers

![Bar chart showing API development and usage by experience]

- Developing APIs
- Using APIs

- Up to 1 year: 9 (Developing), 9 (Using)
- 1 to 3 years: 10 (Developing), 10 (Using)
- 3 to 5 years: 6 (Developing), 6 (Using)
- 5 to 7 years: 7 (Developing), 7 (Using)
- More than 7 years: 4 (Developing), 4 (Using)
Used notations in practice
-38% of respondents-

- 54% of respondents used UML Activity Diagrams
- 85% used UML Sequence Diagrams
- 8% used BPMN Choreography
- 31% used in-house developed notations
- 15% used other standard notations
RESTalk’s intuitiveness

POST /resource <empty>

201 Created
Location: /resource/X

PUT /resource/X <content>

200 OK
The goal of this RESTful conversation is creating a new resource. The client can send the POST request multiple times. By sending multiple POST requests multiple resources are being created. The client knows the link to the created resource before the start of the conversation. Average correct answers.
RESTalk vs. Standard BPMN Choreography

-41% of respondents-

RESTalk

Standard BPMN Choreography
RESTalk vs. Standard BPMN Choreography

-41% of respondents-

![Bar chart showing the comparison between RESTalk and Standard BPMN Choreography in terms of Conciseness, Expressiveness, and Understandability. The chart indicates that 80% of respondents found RESTalk more concise, 60% found it more expressive, and 75% found it more understandable.](image)
Reading task
-Long Running Request-
Reading task
-Long Running Request-

How many resources are created during this conversation: 11%

- The client must delete the job output resource after it reads it: 56%
- You can access the job output without having a link to the job itself: 62%
- What happens when you try to access the job resource while the job has not completed yet: 72%
- The job resource can be deleted without deleting the job output resource: 72%
- The job output resource gets automatically deleted once the client has read it: 77%
- The client can decide to delete the job output resource only after it has read it: 81%
- When can you delete the job resource: 83%
- The client can read the job output multiple times: 85%

Average: 66% → 73%

Correct answers by sector:
86% in academia and 68% in industry
Modeling task
-CRUD operations on a resource-
RESTalk’s evaluation

Positive Sentiment

- Easily understandable: 83%
- Time efficient: 77%
- Concise: 93%

Would use RESTalk – 78%
Would prefer a tool – 69%
Conclusions

Take-Aways

Further existing work

Future work
• **Goal 1:** Evaluate the need in industry for a domain specific language for modeling RESTful conversations

• **Goal 2:** Evaluate the cognitive characteristics of RESTalk

*Qualitative research technique => No statistical inference*
Take away

- Goal 1: Evaluate the need in industry for a DSML-

38% already using some notation
78% willing to use RESTalk
Take away

-Goal 2: Evaluate the cognitive characteristics of RESTalk-

**Intuitiveness:**
77% correct answers without prior RESTalk knowledge

*Feedback:* include the rationale behind server’s decisions at XOR gateways

**Closeness of mapping to the problem world:**
61% found RESTalk more concise than what they are using

*Feedback:* make state transitions and looping limits explicit in RESTalk

**Abstraction gradient:**
83% found RESTalk easy or somewhat easy to understand

*Feedback:* use end events only after a DELETE method and make dependencies among resources explicit
Further Existing Work

RESTalk Extension
- roles
- state transitions
- async email interactions
- multiparty


RESTalk Pattern Language
http://restalk-patterns.org

Future Work

Tool development:
- natural language
- graphical modeling
- code generation

Feedback implementation

RESTalk Use

Survey

Experiment

With statistical relevance

Applied:
- on patterns
- on real APIs
- by practitioners
Contribute to our research on RESTful conversation patterns!
http://restalk-patterns.org/contribute.html

Contact us at:
ana.ivanchikj@usi.ch
c.pautasso@ieee.org
silvia.schreier@innoq.com