RESTful RNA Folding

Michael T. Wolfinger

16 February 2013
Bled, Slovenia

universität wien

Representational State Transfer (REST)

Representational State Transfer (REST) is a software architecture style for distributed systems such as the World Wide Web that has emerged as a predominant Web Service design model.

Initially introduced in Roy Fielding's PhD dissertation (2000).

RESTful Web Services use **HTTP** als the *underlying application protocol*, **URIs** to *locate resources* and (among others) **XHTML**, **XML** or **JSON** get *get a representation*.

REST Basics

RESTful architectures are based on a simple client-server model:

- A client initiates a request to a server
- A server processes the response and returns a response

A key feature of **RESTful** architectures is the fact that requests and responses are built around the transfer of **representations** of different **resources**.

Resource: A coherent entity that can be addressed

Representation: A document that captures the state of a resource

What is a Web Service?

- A web page that is meant for a computer to request and process
- Consumed by an autonomous program rather than by a Web Browser
- Requires *architectural style* (because client is not a smart human)

An architectural style is a *coordinated set of architectural constraints* that restricts the roles and features of architectural elements, and the allowed relationships among those elements, within any architecture that conforms to that style

- A style can be applied to many architectures
- An architecture can consist of many styles

universität wien

The REST model

Traditional distributed systems rely on a shared model (a.k.a. API). A **RESTful** system relies on three properties:

- **Nouns**: Used to name the resources of interest
- **Verbs:** Operations that are applied to named resources ⇒ HTTP methods
- **Content Types:** Used to define resource representations

Nouns in a RESTful System

Nouns are used to name resources, in most designs these will be the URIs
$\Rightarrow$ URI Design

- Anything of interest should be named
- Applications can talk about named things

Separating nouns from verbs and representations improves extensibility

- applications might still work with resources without being able to process them
- introducing new operations on the Web does not break the Web

HTTP Methods in a RESTful System

REST aims at using universal *verbs* (those that can be applied to all nouns) only. **HTTP/1.1** covers eight *verbs* (a.k.a. *methods*, *operations*) that are valid for all resources. For most applications, the following HTTP methods are sufficient:

- `GET`: Fetch a resource's representation
- `PUT`: Create or update a resource
- `POST`: Add to a resource (instead of an overwriting update)
- `DELETE`: Delete a resource

These correspond to the basic **CRUD** (**c**reate, **r**ead, **u**pdate, **d**elete) database operations.

universität wien

RESTful Web Services

Representational State Transfer (REST) is a set of design principles to qualify architectures. As such, **REST** is an architectural style.

- Resources are identified by Uniform Resource Identifiers (URIs)
- Resources are manipulated through their representations
- Messages are self-descriptive and stateless
- Multiple representations are accepted or sent
- Hypertext is the engine of application state

Resource Oriented Architecture (ROA)

- Method information goes into the HTTP method
- Scoping information goes into the URI
- Content type information goes into the HTTP header

RESTful State Management

In a **RESTful** design

- **State** is transferred between server and client
- **State** is represented as part of the transferred content
- Data transfer is **not state-specific** (no stateful connection handling)

Advantages:

- Scalability
- Resources can be made available on different servers
- Clients are not affected by server interruption/failure

universität wien

REST Principles

It is easy to break down REST to five core principles

- Resources with unique Identification
  - Use URIs to identify persistent (individual or set) entities
- Linking / Hypermedia
  - Use hyperlinks to connect resources and direct the application flow
- Standard methods
  - Use standard HTTP methods like GET, PUT, POST, DELETE
- Different representations
  - Use different representations of resources to satisfy requirements
- Stateless Communication
  - Make the client hold any state information

universität wien

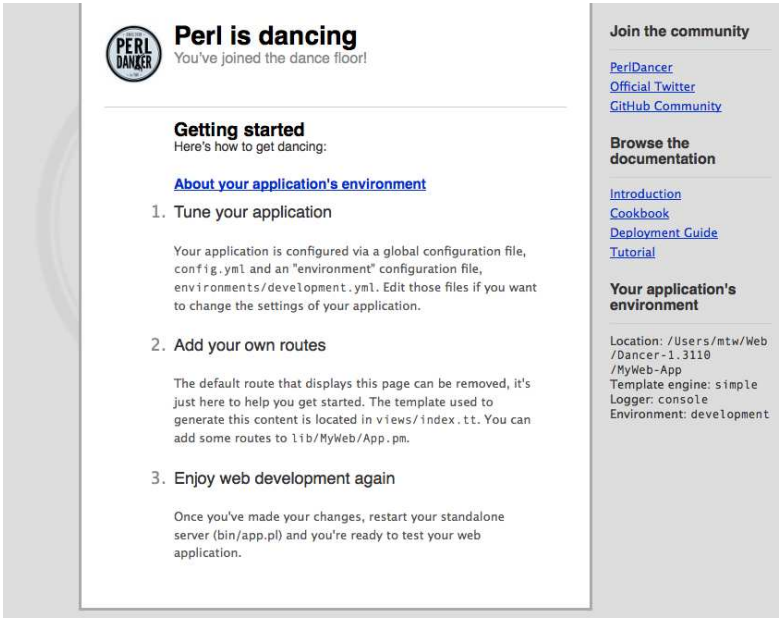
Perl Dancer: A powerful Web application framework

Dancer is a micro-framework for writing RESTful Web applications, inspired by *Ruby's Sinatra* framework. Dancer comes with:

- intuitive, minimalist and dead simple syntax
- a standalone lightweight server
- support for FastCGI and CGI backends
- a simple templating engine plus support for TemplateToolit
- clear separation between dynamic vs. static (JS,CSS,images)
- dedicated pages for various HTTP status codes (500 / 404 etc.)

Perl Dancer in action

```
package MyApp;
use Dancer;
our $VERSION = '0.1';
  get '/' => sub {
    template 'index' ;
  };
true;
```



Perl Dancer Route Handlers

- A perl Dancer application is a collection of *route handlers*
- A route handler is, basically, a *subroutine*
- It is bound to a *HTTP method*
- And a path or *path pattern*

Dancing recipe:

- Choose a *HTTP method*
- Add a *path segment*
- Mix it with a *subroutine*
- Sprinkle *plugins and keywords* on top

Perl Dancer Route Structure

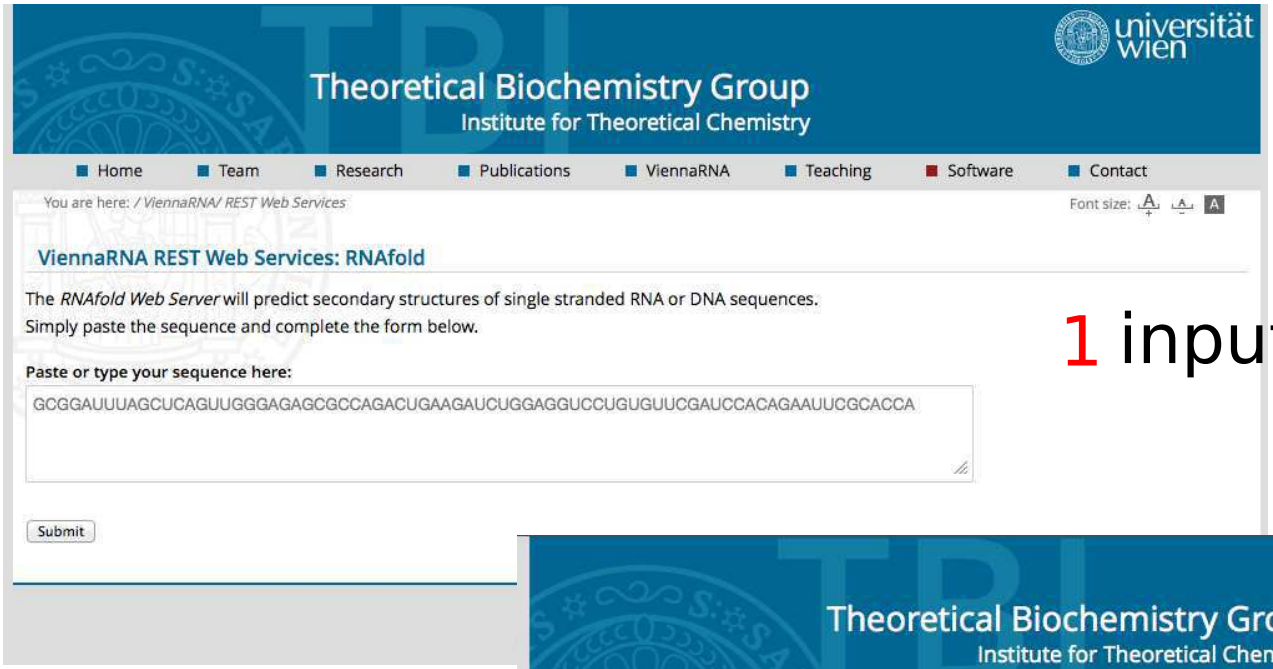
- `get '/path' => sub { ... };`
- `post '/path' => sub { ... };`
- `put '/path' => sub { ... };`
- `del '/path' => sub { ... };`
- `options '/path' => sub { ... };`
- `any '/path' => sub { ... };`

Dynamic Routes

Besides *static routes*, Perl Dancer can handle *dynamic routes*

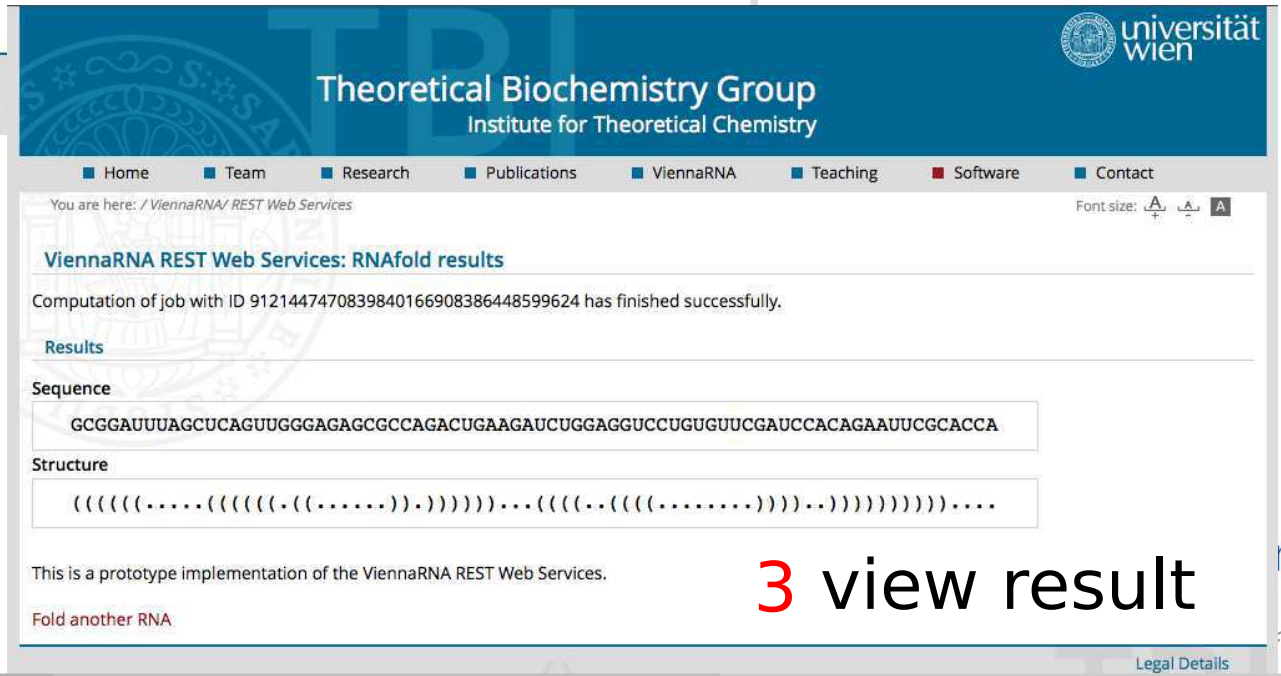
```
get '/sequence/:id' => sub {
  # do something with params->{id}
  ...
};

get '/user/:gid/:uid' => sub {
  # use params->{gid} and params->{uid}
  ...
};

get qr{/ (\w+) / \d{2,3} (.+)? } => sub {
  ...
};
```

RESTfold: A RESTful interface for RNA folding



1 input sequence

2 Ajax magic



3 view result

Scripted Interaction with RESTfold 1

AJAX request

```
curl -i -H "Accept: application/json" -H 'X-Requested-With: XMLHttpRequest'
-X POST -d "sequence=GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGAUCUGGAGGUCCUGUGU
UCGAUCCACAGAAUUCGCACCA" http://rest.tbi.univie.ac.at/fold
```

JSON response

```
HTTP/1.1 200 OK
Date: Sat, 16 Feb 2013 12:09:44 GMT
Server: Apache/2.2.20 (Ubuntu)
X-Powered-By: Perl Dancer 1.3070
Set-Cookie: RNA.session=99041400312760626954; path=/;
expires=Sat, 16-Feb-2013 12:09:55 GMT; HttpOnly
Content-Length: 260
Content-Type: application/json; charset=utf-8
{
   "sid" : "5586904282384",
   "reply_RNA_sequence" : "GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGAUCUGGAGGU
                           CCUGUGUUCGAUCCACAGAAUUCGCACCA",
   "result_link" : "http://rest.tbi.univie.ac.at/fold/data/5586904282384"
}
```

Scripted Interaction with RESTfold 2

AJAX request

```
curl -i -H "Accept: application/json" -H 'X-Requested-With: XMLHttpRequest'
-X GET http://rest.tbi.univie.ac.at/fold/data/5586904282384
```

JSON response

```
HTTP/1.1 200 OK
Date: Sat, 16 Feb 2013 12:10:17 GMT
Server: Apache/2.2.20 (Ubuntu)
X-Powered-By: Perl Dancer 1.3070
Content-Length: 232
Content-Type: application/json; charset=utf-8
{
   "done" : 1,
   "sequence" : "GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGAUCUGGAGGUCCUGUGUUCGA
                 UCCACAGAAUUCGCACCA",
   "structure" :"((((((.....((((((.((......)).))))))...((((..((((........))
                 ))..))))))))))....."
   "energy" : "-21.70"
}
```

RESTfold workflow

1. (C) `POST` sequence and parameters to `http://<domain>/fold`
2. (S) Create persistent reusource for results `http://<domain>/fold/data/<uid>`
3. (S) Execute RNAfold (or queue in Grid Engine)
4. (S) Return JSON object with results resource to client
5. (C) `GET http://<domain>/fold/data/<uid>`
6. (S) Return JSON object with computational results

universität wien

Benefits of RESTful RNA folding Web Services

- ViennaRNA REST API
- Allows for efficient automation
- High scalability (server side)
- Short, non-redundant code for human/machine interface
- No CGI required

universität wien

Summary

- **REST** is an architectural design style for Web Applications
- **REST** heavily relies on ready-to-use HTTP properties
- **RESTful architectures** scale very well with data traffic
- **Perl Dancer** is a new framework for building **RESTful** Web Services
- **RESTfold** is the first step towards a suite of **RESTful** Web Services for RNA folding

universität wien

References

- *REST API Design Rulebook*, Mark Masse, O'Reilly 2011
- *REST und HTTP*, Stefan Tilkov, dpunkt Verlag 2009
- *Service Design Patterns*, Robert Daigneau, Addison-Wesley 2011
- Web Architecture course by Erik Wilde

universität wien

Acknowledgements

- Ronny Lorenz
- Fabian Amman
- RNA-REG SFB
- All TBI and CIBIV folks