Take a small REST

Simple approaches for REST in smalltalk

Norbert Hartl
2denker

What we do...

...at 2denker

- mobile applications
- backend services for mobile applications
- backend services for b2b

Why REST?

- there are RPC-style services (SOAP, XML-RPC,...)

Why REST?

- there are RPC-style services (SOAP, XML-RPC,...)
- REST is data-centric

Why REST?

- there are RPC-style services (SOAP, XML-RPC,...)
- REST is data-centric
- REST/HTTP has meta data

Why REST?

- there are RPC-style services (SOAP, XML-RPC,...)
- REST is data-centric
- REST/HTTP has meta data
- REST is about identity

Smalltalk tools to do REST

- Magritte
- Magritte-XMLBinding
- Magritte-JSON
- Seaside-REST

serialization/materialization

using magritte meta tool support

- model with meta information
- validation of model using meta information
- conversion into/from formats with meta info

Example class Person

- fullName
- emailAddress

model with meta information

```
Person>>#fullNameDescription
    <magritteDescription>
    ^ MAStringDescription new
        accessor: #fullName;
        label: 'full name';
        beRequired;
        priority: 100;
        yourself
```

model with meta information

(using pragma to add a description)

```
Person>>#fullNameDescription
    <magritteDescription>
    ^ MAStringDescription new
        accessor: #fullName;
        label: 'full name';
        beRequired;
        priority: 100;
        yourself
```

model with meta information

(adding type information)

```
Person>>#fullNameDescription
    <magritteDescription>
    ^ MAStringDescription new
        accessor: #fullName;
        label: 'full name';
        beRequired;
        priority: 100;
        yourself
```

model with meta information

(specifying value store/retrieve operation)

```
Person>>#fullNameDescription
    <magritteDescription>
    ^ MAStringDescription new
        accessor: #fullName;
        label: 'full name';
        beRequired;
        priority: 100;
        yourself
```

validation using meta info

(magritte)

```
Person>>#emailAddressDescription
    <magritteDescription>
    ^ MAStringDescription new
        accessor: #emailAddress;
        label: 'email address';
        addCondition: [:val| val includes: $@ ]
           labelled: 'address must contain @';
        beRequired;
        priority: 200;
        yourself
```

validation using meta info

(magritte)

```
Person>>#emailAddressDescription
    <magritteDescription>
    ^ MAStringDescription new
        accessor: #emailAddress;
        label: 'email address';
        addCondition: [:val| val includes: $@ ]
          labelled: 'address must contain @';
        beRequired;
        priority: 200;
        yourself
```

validation using meta info

(magritte)

```
Person>>#emailAddressDescription
    <magritteDescription>
    ^ MAStringDescription new
        accessor: #emailAddress;
        label: 'email address';
        addCondition: [:val| val includes: $@ ]
          labelled: 'address must contain @';
        beRequired;
        priority: 200;
        yourself
```

adding conversion info

(for Magritte-XMLBinding)

```
fullNameXmlDescription: aDescription
    <magritteDescription: #fullNameDescription>
    ^ aDescription xmlElementName: 'fullname'
```

```
emailAddressXmlDescription: aDescription
   <magritteDescription: #emailAddressDescription>
    ^ aDescription xmlAttributeName: 'email'
```

conversion into/from format

```
p := Person  new.
p
    fullName: 'John Doe';
    emailAddress: 'john@doe.it'.
p magritteDescription toXml: p
```

```
<Person email="john@doe.it">
     <fullname>John Doe</fullname>
</Person>
```

REST call flow

- message routing and parameter handling
- content type selection
- serializiation/materialization to/from network data

message routing

```
time
    <get>
    <path: '/time'>

    self requestContext response
        status: 200;
        nextPutAll: Time now greaseString;
        respond
```

message routing

(matching HTTP verb)

```
time
    <get>
    <path: '/time'>

    self requestContext response
        status: 200;
        nextPutAll: Time now greaseString;
        respond
```

message routing

(matching path/parameters)

```
time
    <get>
    <path: '/time'>

    self requestContext response
        status: 200;
        nextPutAll: Time now greaseString;
        respond
```

content type xml

```
getPersonXml: aString
    <get>
    <path: '/person/{1}'>
    <produces: 'application/xml'>

    self requestContext response
        status: 200;
        nextPutAll: (
            self description toXml: (self personWithId: aString));
        respond
```

content type xml

(placeholder in url become method paramter)

```
getPersonXml: aString
    <get>
    <path: '/person/{1}'>
    <produces: 'application/xml'>

    self requestContext response
        status: 200;
        nextPutAll: (
            self description toXml: (self personWithId: aString));
        respond
```

content type xml

(content type selection based on Accept header)

```
getPersonXml: aString
    <get>
    <path: '/person/{1}'>
    <produces: 'application/xml'>

    self requestContext response
        status: 200;
        nextPutAll: (
            self description toXml: (self personWithId: aString));
        respond
```

content type json

```
getPersonJson: aString
    <get>
    <path: '/person/{1}'>
    <produces: 'application/json'>

    self requestContext response
        status: 200;
        nextPutAll: (String streamContents: [:stream|
            (self personWithId: aString) jsonOn: stream]);
        respond
```

example post

```
addPersonFromJson
    <post>
    <path: '/person'>
    <consumes: 'application/json'>

    self addPersonUsing: [ self materializeFromJson ]
```

example post

(content type selection based on Content-Type header)

```
addPersonFromJson
    <post>
    <path: '/person'>
    <consumes: 'application/json'>

    self addPersonUsing: [ self materializeFromJson ]
```

Demo

Thank you!

Questions?