REST e MASHUPs

Como desenvolver web services pensando na integração com outros web services.

Israel Lacerra
Omar Ajoue
Roberto Bodo
Thiago Coraini

REST

1. Introdução

Com a popularização dos web services, diversas companhias iniciaram o desenvolvimento de aplicações web das mais diversas formas. Como toda nova tecnologia, é difícil que esta surja já com um padrão bem definido sobre como estruturar suas aplicações.

Roy Fielding, co-fundador da Apache Software Foundation definiu o termo web services em seu trabalho de Ph.D. em 2000, e propôs um estilo arquitetural sobre como criar web services.

A sigla significa REppresentational State Transfer, ou Transferência de Estados Representacionais. A idéia é que tenhamos algo muito semelhante à estrutura da Web, em que URLs indicam estados, e que o usuário navegue para outros estados por meio de ligações explicitadas no estado anterior.

2. Estados

Um estado, pelo estilo arquitetural REST, é um endereço (URL) que representa uma informação que o web service oferecerá. Podemos citar, como exemplos de estados, endereços do tipo:

```
http://www.burgerking.com.br/cardapio
```

Cada "pasta" é na verdade um mapeamento lógico para um nível diferente de navegação que podemos seguir. A idéia assemelha-se à navegação em uma árvore, em que temos nós e nós filhos. A resposta de um estado é dada por uma resposta a uma requisição HTTP, que pode vir tanto em formato XML, JSON ou até texto puro. Podemos ter como resposta a um estado um xml da forma:

```
<?xml version="1.0"?>
<p:Cardapio xmlns:p="http://www.burgerking.com.br/cardapio"
xmlns:xlink="http://www.w3.org/1999/xlink">
<Tipo nome="Sobremesas"
     xlink:href="http://www.burgerking.com.br/cardapio/sobremesas"/>
<Tipo nome="Sanduiches"
     xlink:href="http://www.burgerking.com.br/cardapio/sanduiches"/>
<Tipo nome="Combos"
```

```
     xlink:href="http://www.burgerking.com.br/cardapio/combos"/>
</p:Cardapio>
```

Ou seja, para cada estado, temos um conjunto de dados que nos indicam informações a respeito do estado atual e nos dão direções sobre como prosseguir a outros estados. Conforme exemplo acima, não temos nenhuma informação a respeito do estado atual além das informações de navegação.

Estes estados representam, no estilo arquitetural REST, os recursos que o seu web service oferecerá aos clientes.

3. Resource Oriented Architeture - ROA

Os recursos, representados logicamente pelas URLs de estado, indicam o tipo de informação que o seu Web Service oferecerá. No exemplo citado acima, nosso recurso inicial apresentado ao cliente é o cardápio, que abre a navegação para outros recursos diferentes: Sobremesas, sanduíches e combos. Qualquer um destes novos recursos pode tanto informar dados sobre si mesmo como apresentar novas estruturas para navegar.

Em REST, a idéia é que as informações sejam apresentadas ao cliente passo a passo, permitindo que o mesmo navegue pela estrutura apresentada. Jamais dê todas as informações logo numa primeira requisição. Visa-se a otimização da utilização de banda. Este é um dos pilares do estilo arquitetural REST: cache de informações e economia nas transferências.

Criando um novo exemplo para ilustrar a idéia dos recursos:

`http://blog.exemplo.com/posts`
Representa o recurso posts do web service relacionado ao nosso blog.

`http://blog.exemplo.com/posts/515`
Representa a instância 515 do recurso posts em nosso blog.

`http://blog.exemplo.com/posts/515/comments`
Representa os comentários referentes à instância 515 dos nossos posts do blog.

E exemplificando o segundo recurso apresentado acima, da instância 515 do recurso posts, poderíamos ter uma resposta da forma:

```
<?xml version="1.0"?>
<p:Post xmlns:p="http://blog.exemplo.com"
xmlns:xlink="http://www.w3.org/1999/xlink">
<Post-ID>515</Part-ID>
<Titulo>Teste de Post</Titulo>
<Texto>Isto é o conteúdo do post!</Texto>
<Comentarios
    xlink:href="http://blog.exemplo.com/posts/515/comentarios"
  />
<Autor>Anônimo</Autor>
</p:Post>
```

Ou seja, os estados (URLs) representam recursos que o nosso sistema apresentará. O conceito vai um pouco de encontro ao que é mais comumente visto na programação, em que apresentam-se verbos de ação para os endereços. É comum que vejamos web services da forma `http://www.exemplo.com/rest?method=getComments&parameter=515`. Apesar de correto, o recurso mostrado não segue o estilo arquitetural REST à risca, pois não utilizamos recursos da forma como é esperado pelo estilo arquitetural REST e foram enviados parâmetros, inclusive com o verbo de ação (getComments, no caso) como sendo parte da mensagem.

4. Verbos de ação

No estilo arquitetural REST, prega-se que utilizemos como identificadores de ação os verbos disponibilizados pelo HTTP. Além dos já conhecidos GET e POST, devem-se utilizar os verbos PUT, DELETE e HEAD.

GET: Verbo utilizado para solicitar informações. Deve ser livre de efeitos colaterais no lado servidor.
POST: verbo para inserção de novas informações. Utilizado para envio de fato do conteúdo.
PUT: verbo utilizado para atualização de dados, ou dependendo da implementação, pode ser um híbrido de inserção em caso de registro novo e atualização em caso de registro existente.
DELETE: verbo utilizado para indicar remoção de informações.
HEAD: verbo no estilo "ping", que serve para obter informações sobre a existência de um recurso sem obter todas as informações a respeito do recurso.

No estilo arquitetural REST, os verbos HTTP são mandatários no tipo de ação que a requisição fará. Evite URLs da forma: /rest?method=package.class.method()&param1=1&param2=b. Esta é uma estrutura para chamadas a métodos, e não a recursos, como é a idéia do REST.

5. Balanço e conclusão sobre REST

REST, como apresentado acima, não apresenta nenhum novo recurso. Utiliza o já existente HTTP e os já conhecidos XML ou JSON para representar os recursos devolvidos pela aplicação. Possui então uma interface uniforme, em que pensamos nos verbos HTTP para indicar o tipo de ação, e os recursos para evidenciar o tipo de informação que é requisitada. A navegação entre os recursos é bastante intuitiva, pois é explicitada a cada estado navegado.

REST é criticado por muitos por ser considerado simplista demais: dizer que suas aplicações apenas fazem inserções, remoções, atualizações e consultas a bancos de dados é pequeno demais, comparado à complexidade de aplicações maiores.

Para aplicações web em geral, web services especificamente, esta abordagem é suficiente.

MASHUP

1. Introdução

O termo "mashup" está relacionado com o ato de unir informações de diversas fontes. No contexto que nos interessa, podemos dizer que um Mashup é um site ou uma aplicação web que é composta da união de funcionalidades e conteúdo pré-existentes.

Exemplos:

1. *chicagocrime.org*
   Esse site é o mais mencionado em textos sobre o assunto, por ter sido um dos primeiros a utilizar um mashup. Basicamente, o site reúne estatísticas da cena criminal de Chicago e utiliza o Google Maps para mostrar as regiões mais perigosas.
2. *tv.timbormans.com*
   Site que reúne funcionalidades do Lastfm e funcionalidades do Youtube. Basicamente, ao tocar uma playlist do Lastfm, podemos visualizar o vídeo da música tocada no momento, através do Youtube.
3. *flickrvision.com*
   Esse site mostra as últimas atualizações feitas por usuários do Flickr e, utilizando o Google Maps, exibe em qual cidade essas atualizações foram feitas.

2. Como fazer um Mashup?

Para criarmos um Mashup temos que resolver alguns problemas básicos. Abaixo temos uma lista de cinco deles:

1. recuperação de dados;
   A principal idéia de um Mashup é reutilizar dados já existentes. Portanto, precisamos coletar esses dados de alguma forma.
   Em alguns casos, podemos utilizar algum provedor de conteúdo que já retorna os dados todos organizados em um .xml, por exemplo.
   Em outros, podemos ter um pouco mais de trabalho. Por exemplo, podemos ter um .html em mãos e devemos usar expressões regulares para coletar alguma informação dele.
2. modelagem dos dados;
   Após extrair os dados, podemos querer classificar esses dados conforme nossas necessidades. Se estivermos criando alguma ferramenta, que faz um mashup automatizado, esse item é de extrema importância.
3. limpeza dos dados;
   Podemos encontrar dados que não nos interessam ou até mesmo dados em algum formato diferente do que queremos. Logo, devemos descartar alguns dados e padronizar o formato de outros.
4. integração dos dados;
   Após coletarmos e normalizarmos tudo o que precisamos, precisamos reunir todos os dados na nossa aplicação (ou nosso site) de alguma forma.
5. visualização final.
   Nesse item decidimos como vamos mostrar os dados integrados para o usuário. Vamos utilizar mapas, tabelas, gráficos? Como o usuário interage com o site?

3. Popularidade

Nos últimos anos, a popularidade dos Mashups vem crescendo. Mas quais são os motivos? Abaixo apresentamos dois motivos principais:

1. aumento do número de provedores de conteúdo;
   Junto do crescimento do número de sites que utilizam mashup, temos o crescimento do número de sites que fornecem o seu conteúdo para mashup. Sites como Google Maps, Flickr, Amazon, Youtube fornecem algumas de suas funcionalidades e quase todo seu conteúdo, de forma organizada, para usuários que queiram fazer um mashup.
2. criação de editores para Mashup.

Nos últimos anos foram criados diversos editores, voltados para o usuário comum da internet. Com interface gráfica amigável e muito fáceis de usar, possibilitam que qualquer pessoa faça seu próprio mashup. Entre os editores mais utilizados temos: Popfly (Microsoft) e o Pipes (Yahoo).

4. Mashups + REST

O estilo arquitetural REST possibilita a criação de Mashups de forma mais elegante. No exemplo mostrado no item 2, do texto sobre REST acima, temos o site do Burguer King, que nos fornece dados sobre seu cardápio, com uma organização extrema. Se desejarmos fazer um site de delivery de fast food, por exemplo, podemos utilizar os arquivos .xml para extrair as informações que desejamos e utilizá-las em nosso site.

Podemos verificar inclusive, pelo site *programmableweb.com*, que mais de 60% das APIs utilizadas para criar mashups, utilizam REST.