# Um Estudo Comparativo das Aplicações Web 2.0 visando a Integração Automática de seus Serviços

Maurício N C Bomfim, Fabio Ferrentini Sampaio, João Sérgio S Assis
NCE-PPGI/UFRJ

Caixa Postal 68.530 – 21.945-970 – Rio de Janeiro – RJ – Brasil
{mauricio,ffs,joao}@nce.ufrj.br

**Abstract.** One of the main characteristics of Web 2.0 applications is the ability to merge their features creating mashups. The automatization of this process, however, is hampered by the lack of standardization of the APIs provided by these applications. This paper presents a framework that organizes the main aspects of componentization in Web 2.0 applications according to their level of adherence to the REST architectural style and its mechanisms of authentication. We held an exploratory study comparing some existing APIs and classifying them according to these criteria. The findings of this study may serve as a basis for development of environments that are proposed to automate the integration of these applications.

**Keywords:** Web2.0, REST, autenticação, *mashups*.

## 1 Introdução

A nova geração de aplicativos da Web conhecida como Web 2.0 [13], tem influenciado um número cada vez maior de serviços disponíveis na Internet. Uma de suas principais características é a possibilidade de mesclar funcionalidades de aplicações diferentes utilizando APIs (Application Programming Interfaces). As APIs das aplicações Web 2.0 nada mais são que coleções de recursos e métodos acessados através de requisições HTTP. Os métodos de uma API possuem conjuntos de parâmetros e uma ou mais possibilidades de formatos de saída, como XML ou JSON [6]. É possível criar novas aplicações, conhecidas como *mashups*, a partir de chamadas às APIs de outras aplicações já existentes, incorporando assim seus dados e funcionalidades.

A principal tecnologia que apóia este modelo de componentização de Web 2.0 é o REST - *Representational State Transfer* [7,8], um estilo de arquitetura de software para sistemas hipermídia distribuídos que se aplica perfeitamente ao funcionamento do protocolo HTTP e conseqüentemente, da Web. REST é implementado como um conjunto de critérios e práticas de projeto utilizadas com o intuito principal de obter escalabilidade dos sistemas de software projetados à luz de seus preceitos. O REST vem cada vez mais se tornando uma tendência no desenvolvimento de serviços e aplicações na Web 2.0.

O desenvolvimento de *mashups* é uma tarefa não trivial que exige conhecimentos de programação através de linguagens de *script*. Algumas soluções, como a utilização de ambientes para criação de *mashups* ou de plataformas para agregação de *Web Widgets*, embora se destinem a facilitar o trabalho para usuários finais, não são ferramentas suficientemente genéricas. Nestes casos, o usuário está limitado à utilização das *Widgets* existentes ainda que estas plataformas geralmente ofereçam APIs para facilitar o processo de desenvolvimento de novas *Widgets*.

Com o intuito de oferecer ao usuário a possibilidade de incorporar de maneira simples – e sem a necessidade de programação – qualquer aplicação externa que possua uma API aberta, a abordagem que foi explorada neste trabalho é a da integração automática. Embora se busquem padrões que permitam facilitar o processo de integração destes componentes, ainda existem muitas iniciativas proprietárias isoladas, o que dificulta a realização desta tarefa.

Assim, para desenvolver ferramentas que automatizem a integração de aplicações, é necessário resolver os seguintes problemas: a existência de APIs que não são totalmente aderentes aos preceitos REST e a falta de padronização no processo de autenticação. Neste sentido, este trabalho é um estudo exploratório que se propõe a analisar até que ponto as APIs das aplicações da Web 2.0 são aderentes ao estilo de arquitetura REST, bem como os mecanismos de autenticação utilizados. Os resultados obtidos a

partir desta análise podem ser de grande valia para a proposição de um modelo que facilite e automatize a integração de aplicações externas.

Na Seção 2 apresentamos o estilo de arquitetura REST assim como os preceitos que devem ser observados na implementação de um sistema aderente a este estilo. Na Seção 3 fazemos considerações sobre os principais aspectos relativos à autenticação de usuários nestas aplicações. A Seção 4 propõe a utilização de uma rede sistêmica[1] como *framework* para a classificação das APIs, levando em conta o grau de adesão destas à arquitetura REST e o modelo de autenticação utilizado. Na Seção 5 aplicamos esta proposta de comparação a um conjunto de APIs existentes de forma a determinar o seu comportamento. Finalizamos com a apresentação das conclusões deste trabalho na Seção 6.

## 2 O Estilo de Arquitetura REST

O conceito de recurso é a principal abstração do estilo de arquitetura REST. Um recurso é qualquer informação relevante para o sistema, que precise ser referenciada através do protocolo. Para identificação dos recursos, é necessário um espaço unificado de nomes, e para isso, a sintaxe universal utilizada é a URI. Cada recurso deve ter uma URI associada a ele, assim como cada URI deve estar associada a um único recurso.

Representação de um recurso é o conjunto de dados que define as informações associadas ao estado de um recurso. Utiliza para isso um formato padronizado que pode ser, por exemplo, um arquivo XML, uma página Web, uma string JSON, ou simplesmente um arquivo texto com seqüência de valores separados por vírgulas. Ao requisitar um recurso, o que se obtém é uma representação do mesmo. Da mesma forma, ao incluir ou alterar um recurso envia-se ao servidor uma representação.

Para determinar as operações realizadas sobre os recursos, aplicações REST devem utilizar-se de interfaces uniformes, ou seja, um conjunto restrito de operações com significados bem definidos. O protocolo HTTP implementa através dos métodos GET, POST, PUT e DELETE as operações básicas de consulta, inclusão, alteração e exclusão. Sendo assim, é natural que a utilização de REST sobre HTTP utilize-se destes métodos sempre que referenciar um recurso. Uma URI, quando requisitada através de um verbo HTTP, implica na execução de um método, cujas informações sobre o escopo de sua execução devem ser mantidas na própria URI. A grande vantagem do uso de interfaces uniformes decorre do fato de a Internet ser uma rede composta não apenas de clientes e servidores, mas possuir elementos intermediários como *firewalls*, roteadores e *proxies*, responsáveis respectivamente por aumentar a segurança, prover escalabilidade e melhorar o desempenho da rede de uma maneira geral. O uso de uma semântica padronizada permite sua interpretação por estes elementos otimizando seu funcionamento.

REST é um protocolo cliente/servidor sem estado: cada requisição HTTP deve conter todas as informações necessárias para a sua execução, pois ela deve ser completamente independente das demais. Se o servidor, por alguma razão possui estado, este também deve ser encarado como um recurso, de forma que tenha sua própria URI. Qualquer informação de estado deve permanecer no cliente, sendo transmitida ao servidor a cada requisição que a necessite.

As representações dos recursos devem ser interligadas, de forma a permitir a navegabilidade entre eles, alterando assim o estado da aplicação a cada novo recurso requisitado. Este conceito foi chamado por Fielding (2000) de "hipermídia como um mecanismo de estado da aplicação" e significa que as representações devem ser projetadas como documentos hipermídia possuindo ligações com outros recursos. Ao seguir uma dessas ligações efetua-se uma transição de estado da aplicação. Como resultado, é possível navegar a partir de um recurso REST para os demais, seguindo as ligações existentes nas representações de recursos obtidas.

1 Conforme sugerido por BLISS et al. [5], Redes Sistêmicas são formas gráficas de se fazer uma representação resumida de um determinado conhecimento. Trata-se de um instrumento de pesquisa que, a exemplo dos Mapas Conceituais, vem sendo cada vez mais utilizado entre os pesquisadores da área educacional para análise de dados qualitativos. Sua origem, segundo esses autores, vem da lingüística sistêmica ou da também chamada sociolingüística.

# 3 Autenticação de Usuários

Outra característica que diferencia as APIs é a maneira com que seus usuários são autenticados. Autenticação é o processo de identificação da identidade de um usuário que acessa um determinado sistema. Considerando que as aplicações Web são baseadas num protocolo sem estado, a autenticação deve prover uma maneira de determinar qual usuário é responsável por cada requisição realizada.

Uma vez que um usuário tenha sido autenticado, é necessário determinar quais as operações permitidas a ele. Autorização é o nome dado a este processo, de determinar se uma requisição pode ou não ser realizada por um dado usuário.

Embora o processo de autenticação não esteja diretamente relacionado com o grau de aderência de uma API à arquitetura REST, vale à pena estudá-lo, pois qualquer solução proposta para integração automática de aplicações deve levá-lo em consideração.

Neste sentido, os aspectos mais relevantes a serem considerados são: o uso de criptografia e de chaves de autenticação; a forma de envio das informações de autenticação; o modelo utilizado.

## 3.1. Criptografia

A autenticação exige o envio de dados sigilosos como contas e senhas. Para garantir a segurança das informações que trafegam na rede, as APIs normalmente provêm alguns mecanismos que envolvem criptografia. Existe uma forma padrão de fazer isso que é a utilização de SSL [14]. Em alguns casos, quem implementa os serviços desenvolve seu próprio mecanismo de criptografia.

## 3.2. Chaves de Autenticação

Algumas APIs exigem que as aplicações clientes sejam cadastradas previamente para poderem fazer uso de suas funcionalidades. Neste caso, o desenvolvedor precisa obter uma chave (*API Key*) através da qual a aplicação cliente se identifica para a API. Este procedimento não exclui a necessidade do usuário também se identificar através de um dos processos de autenticação descritos anteriormente.

## 3.3. Forma de Envio das Informações

Dentro do escopo deste trabalho, identificamos duas formas principais para o envio de informações de autenticação: utilizando um cabeçalho HTTP ou parâmetros de um formulário.

## 3.4. Modelos de Autenticação

Modelos de autenticação são definidos com o intuito de resolver os problemas de autenticação e autorização descritos acima. Identificamos quatro modelos freqüentemente utilizados nas APIs analisadas.

**Implementadas pelo protocolo HTTP.** O protocolo HTTP provê mecanismos padronizados de autenticação que podem ser utilizados com esta finalidade como, o *HTTP Basic Authentication* e o *HTTP Digest* [9]. A comunicação das informações de autenticação entre cliente e servidor, é realizada através de cabeçalhos HTTP *Authorization* e *WWW-Authenticate*.

Como o protocolo HTTP não mantém estado, e o servidor não tem como associar requisições provenientes de um mesmo usuário, os procedimentos de autenticação HTTP exigem que as credenciais do usuário sejam re-enviadas a cada nova requisição.

Como a codificação em Base64 é um procedimento reversível, este processo não garante a segurança da conta e senha enviadas quando *Basic Authentication* for utilizada. Uma solução possível para este problema é a utilização de SSL/HTTPS, criptografando assim toda a comunicação entre o cliente e o servidor. Outra possibilidade é utilizar o *Digest Autenthication*, outro modelo de autenticação implementado pelo protocolo HTTP onde os dados enviados do cliente para o servidor já são criptografados, garantindo assim a segurança do processo.

**Baseadas em HTTP.** Existem ainda soluções equivalentes que utilizam os mesmos cabeçalhos HTTP (*WWW-Authenticate* e *Authorization*) para implementar algoritmos próprios de autenticação. Nestes casos, assim como na autenticação HTTP, o servidor não mantém estado, sendo necessário reenviar o pedido de autenticação a cada nova requisição. Um exemplo deste tipo de autenticação é a utilizada pelo serviço *Simple Storage Service* da Amazon[2].

**Baseadas em *authentication token*.** Outra solução freqüentemente utilizada em serviços que mantêm estado entre requisições, é a geração de uma chave de identificação de sessão pelo servidor (API-Hash). Esta chave é obtida numa primeira requisição responsável por iniciar uma sessão, devendo então ser devolvida pelo cliente a cada nova requisição ao servidor. A devolução da chave de identificação pode dar-se através de cabeçalhos HTTP ou de um parâmetro de formulário numa requisição POST.

**Soluções onde o usuário não precisa fornecer sua conta e senha para o cliente.** Nestas soluções, o cliente redireciona o usuário para o provedor do serviço para que este solicite diretamente a senha ao usuário e faça a autenticação. É devolvida ao cliente uma chave que permite a realização de novas requisições. Existem vários exemplos de uso deste tipo de abordagem como: o OAuth[3], o Google AuthSub[4], o FlickrAuth[5], o AOL OpenAuth[6] e o Yahoo BBAuth[7].

OAuth é uma tentativa de padronização dos processos de autenticação e autorização para APIs na Web. É um protocolo aberto para autenticação, que permite que sítios Web, ou aplicações ditas Consumidoras (*Consumers*), possam acessar recursos protegidos de um serviço Web ditos Provedores do Serviço (*Service Providers*) através de sua API, sem que seja necessário que os usuários forneçam suas senhas às aplicações Consumidoras.

## 4 Rede Sistêmica para Classificação das APIs

A principal alternativa ao uso de REST na definição de APIs é o RPC (*Remote Procedure Call*) [12], um protocolo para chamada remota de procedimentos. A maior diferença conceitual entre REST e RPC é o fato de RPC ser baseado na diversidade de operações do protocolo, enquanto que REST baseia-se na diversidade de recursos. O SOAP (*Simple Object Access Protocol*) [11] é um exemplo de protocolo do tipo RPC, proposto pela W3C, utilizado com freqüência na definição de Serviços Web. Apesar das aparentes vantagens existentes em seguir a arquitetura REST, muitas APIs não o fazem completamente, levando à criação de sistemas híbridos.

De acordo com a aderência das APIs às arquiteturas REST ou RPC, estas podem ser consideradas como representativas de uma das seguintes classificações propostas por Richardson e Ruby [15]:

- **RESTful** – Totalmente aderente à arquitetura REST. Suas principais características são: expõe os recursos da aplicação através de URIs (*Uniform Resource Identifiers*) [1] e utiliza uma interface uniforme (veja a seção 4.1).
- **Híbrida REST-RPC** – Não pode ser considerada totalmente REST nem totalmente RPC, incorporando simultaneamente características destas duas arquiteturas.
- **Estilo RPC** – Totalmente aderente à arquitetura RPC. Suas principais características são: a utilização de um único endereço para realização das requisições; a utilização de envelopes de dados que são enviados entre o cliente e o servidor; e a definição de um novo vocabulário para especificar cada operação a ser realizada através da API.

A rede sistêmica apresentada na Figura 1 organiza as principais características que diferenciam as APIs das aplicações Web 2.0 analisadas. Estas características são discutidas nas seções 4.1 e 4.2, a seguir.

---

[2] http://aws.amazon.com/s3
[3] http://oauth.net
[4] http://code.google.com/apis/gdata/auth.html
[5] http://www.flickr.com/services/api/auth.spec.html
[6] http://dev.aol.com/openauth
[7] http://developer.yahoo.com/auth/

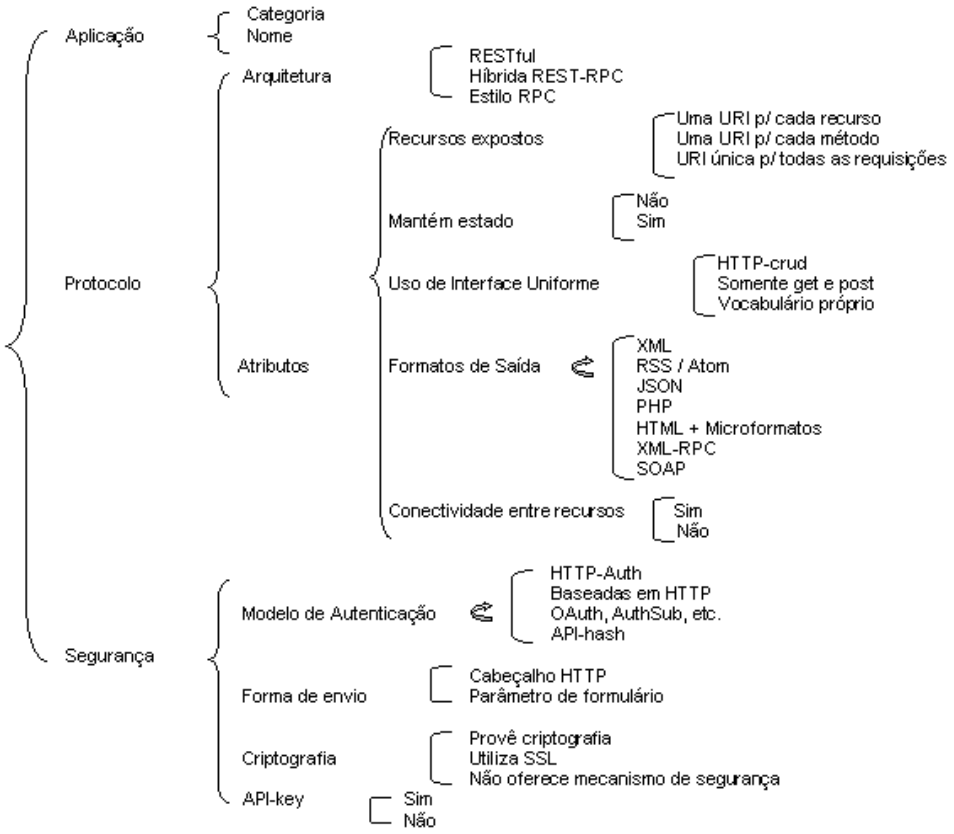


**Fig. 1.** Rede sistêmica das APIs da Web 2.0.

### 4.1. Classificação quanto ao Protocolo

Cinco critérios foram utilizados para comparar as APIs, permitindo enquadrá-las numa das três classificações vistas anteriormente. São eles: os recursos devem ser expostos através de URIs, as requisições não mantêm estado, uso de interface uniforme, formatos utilizados para representação dos recursos e conectividade entre os recursos.

A Tabela 1 mostra cada um destes critérios e algumas soluções freqüentemente adotadas na definição de APIs. Nesta tabela, a numeração (1), (2) e (3) refere-se respectivamente à classificação REST, Híbrida e RPC, conforme foi descrito na seção 2.

**Tabela 1.** Características das APIs REST.

| Critérios | Possibilidades |
|---|---|
| 1. Recursos expostos através de URIs | (1) As informações para identificação do recurso são incluídas na URI (no *path* ou na *query_string*). Utilização de uma URI para cada recurso.<br>(2) As informações para identificação do recurso são incluídas na URI (no *path* ou na *query_string*). Utilização de uma URI para cada método.<br>(3) As informações para identificação do recurso são incluídas no corpo da mensagem. Utilização de uma única URI para todas as requisições. |
| 2. Serviços REST não mantêm estado | (1) Não mantém estado entre requisições<br>(2) ou (3) Mantém estado entre requisições |
| 3. Uso de uma interface uniforme | (1) Uso dos métodos HTTP (GET, POST, PUT, DELETE).<br>(2) Uso dos métodos HTTP (GET para *consultar* e POST para *criar, atualizar e remover*).<br>(3) Uso de um vocabulário próprio (search, getTags, etc) |
| 4. Representações de recursos (formatos de saída) | (1) XML<br>(1) RSS / Atom<br>(1) JSON<br>(1) PHP<br>(1) HTML + Microformatos<br>(3) XML-RPC<br>(3) SOAP |
| 5. Conectividade entre recursos | (1) Existe conectividade entre os recursos<br>(2) e (3) Não existe conectividade entre os recursos |

### 4.2. Classificação quanto à Autenticação

A Tabela 2 apresenta as soluções utilizadas com maior freqüência para cada um dos aspectos relativos a autenticação mencionados na seção 3.

**Tabela 2.** Características das APIs com relação a seu modelo de autenticação.

| Aspectos | Possibilidades |
|---|---|
| Modelo de autenticação | HTTP-Auth<br>Baseadas em HTTP<br>Chave de identificação de sessão (API-Hash)<br>OAuth, Google AuthSub ou equivalente |
| Forma de envio das informações de autenticação | Utiliza o cabeçalho HTTP Authorization<br>Utiliza parâmetros de um Form |
| Criptografia | Provê criptografia<br>Utiliza SSL<br>Não oferece mecanismo de segurança |
| API-key | Utiliza API Key<br>Não utiliza API Key |

## 5 Estudo das APIs da Web 2.0

A fim de determinar o comportamento geral das principais APIs, elegeram-se algumas aplicações populares, às quais foram aplicados os critérios apresentados anteriormente nas Tabelas 1 e 2.

Quanto à determinação dos estilos de arquitetura de cada API, decidiu-se utilizar a seguinte convenção: aquelas com todas as respostas marcadas com (1) são totalmente REST; aquelas com todas as respostas marcadas com (3) são totalmente RPC; qualquer outra combinação indica que a API é híbrida.

Na análise das APIs quanto à autenticação, foram excluídas as versões SOAP e XML-RPC uma vez que estamos interessados em comparar apenas o procedimento de autenticação das APIs ditas REST.

Como resultado deste estudo, a Tabela 3 apresenta os resultados obtidos na classificação das APIs quanto ao estilo de arquitetura e a Tabela 4 sistematiza os processos de autenticação utilizados.

**Tabela 3.** Análise da conformidade com arquitetura REST de algumas APIs.

| Aplicação | | | Protocolo | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | arquitetura | | | atributos | | | | | | | | | |
| | | | | | | 1 | | | 2 | 3 | | 4 | | 5 | |
| | | | RESTful | Híbrida | Estilo RPC | uma URI por recurso | uma URI por método | uma única URI | não mantém estado | get, post, put e delete | vocabulário próprio | Formatos REST | Formatos RPC | Conectividade |
| Favoritos compartilhados | del.icio.us[8] | | | √ | | | √ | | √ | | √ | √ | | |
| | ma.gnolia[9] | | | √ | | | √ | | √ | | √ | √ | | |
| | simpy[10] | | | √ | | | √ | | √ | | √ | √ | | |
| Gerenciamento de tarefas | voo2do[11] | | | √ | | | √ | | | | √ | √ | | |
| | Google Calendar[12] | | √ | | | √ | | | √ | √ | | √ | | √ |
| Repositório de arquivos | Amazon S3[13] | rest | √ | | | √ | | | √ | √ | | √ | | |
| | | soap | | | √ | | | √ | √ | | √ | | √ | |
| | box.net[14] | rest | | √ | | | √ | | | | √ | √ | | |
| | | xml-post | | | √ | | | √ | | | √ | | √ | |
| | | soap | | | √ | | | √ | | | √ | | √ | |
| Repositório de imagens | Flickr[15] | rest | | √ | | | √ | | √ | | √ | √ | | |
| | | xml-rpc | | | √ | | | √ | √ | | √ | | √ | |
| | | soap | | | √ | | | √ | √ | | √ | | √ | |
| | Google Picasa[16] | | √ | | | √ | | | √ | √ | | √ | | √ |
| Wiki | PBwiki[17] | | | √ | | | √ | | | | √ | √ | | |
| Blog | Google Blogger[18] | | √ | | | √ | | | √ | √ | | √ | | √ |
| | Pownce[19] | | √ | | | √ | | | √ | √ | | √ | | √ |
| Repositório de vídeos | Google YouTube[20] | | √ | | | √ | | | √ | √ | | √ | | √ |
| | AOL Truveo[21] | | | √ | | | √ | | √ | | √ | √ | | |
| | AOL Video Upload[22] | | | √ | | √ | | | √ | √ | | √ | | |
| Conteúdo Cultural | BBC[23] | | | √ | | | √ | | √ | | √ | √ | | |

[8] http://del.icio.us/help/api/
[9] http://wiki.ma.gnolia.com/Ma.gnolia_API
[10] http://www.simpy.com/doc/api/rest
[11] http://voo2do.com/help/api
[12] http://code.google.com/apis/calendar/
[13] http://www.amazon.com/gp/browse.html?node=16427261
[14] http://dev.box.net/
[15] http://www.flickr.com/services/api/
[16] http://code.google.com/apis/picasa/
[17] http://api.pbwiki.com/
[18] http://code.google.com/apis/blogger/
[19] http://pownce.pbwiki.com/API+Documentation
[20] http://www.youtube.com/dev
[21] http://developer.truveo.com/index.php
[22] http://dev.aol.com/video/upload
[23] http://www0.rdthdo.bbc.co.uk/services/

**Tabela 4.** Análise do processo de autenticação de algumas APIs.

| Aplicação | | Segurança | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Modelo de autenticação | | | | | | | | Autorização | | Criptografia | | | |
| | | | | Tipo OAuth | | | | | | | | | | | |
| | | HTTP-Auth | Baseada em HTTP | OAuth | AuthSub | flickr | box.net auth | aol | API-Hash | HTTP-Header | Form-encoded | provê criptografia | utiliza SSL | não provê segurança | API-key |
| Favoritos compartilhados | del.icio.us | √ | | | | | | | | √ | | | √ | | |
| | ma.gnolia | | | √ | | | | | √ | √ | | √ | | | |
| | simpy | √ | | | | | | | | √ | | | | √ | |
| Gerenciamento de tarefas | voo2do | | | | | | | | √ | | √ | | | √ | |
| | Google Calendar | | | | √ | | | | | √ | | √ | | | √ |
| Repositório de arquivos | Amazon S3 | | √ | | | | | | | √ | | √ | | | |
| | box.net | | | | | | √ | | | | √ | √ | | | √ |
| Repositório de imagens | Flickr | | | | | √ | | | | | √ | √ | | | √ |
| | Google Picasa | | | | √ | | | | | √ | | √ | | | √ |
| Wiki | PBwiki | | | | | | | | | | | | | | √ |
| Blog | Google Blogger | | | | √ | | | | | √ | | √ | | | √ |
| | Pownce | √ | | √ | | | | | | √ | | √ | | | √ |
| Repositório de Vídeos | YouTube | | | | √ | | | | | √ | | √ | | | √ |
| | AOL Truveo | | | | | | | √ | | | √ | √ | | | √ |
| | AOL Video Upload | | | | | | | √ | | | √ | √ | | | √ |
| Conteúdo Cultural | BBC | Read Only - Não tem autenticação | | | | | | | | | | | | | |

A partir desta análise, foi possível perceber que o uso de REST é uma tendência, mas sua utilização se dá de forma não padronizada. A maior parte das APIs que se dizem REST é, na verdade, híbrida. As principais situações em que os preceitos REST são desrespeitados são os seguintes:

- Exposição de métodos em lugar de recursos;
- Manutenção de estado para simplificar o processo de autenticação;
- Utilização de um vocabulário próprio em vez de utilizar uma interface uniforme;
- Ausência de conectividade entre os recursos.

Como alguns firewalls não permitem o tráfego de operações que utilizam os métodos PUT e DELETE, algumas APIs simplificam a utilização da interface uniforme restringindo-se aos verbos GET para operações seguras (consultar) e POST para operações inseguras (criar, atualizar e remover). Nestes casos é necessário enviar uma informação adicional às requisições do tipo POST para informar qual a real operação a ser realizada, como o cabeçalho *X-HTTP-Method-Override* utilizado pelas APIs da Google[24] ou o parâmetro adicional (por exemplo, method=PUT) utilizado pelas aplicações desenvolvidas no ambiente Ruby on Rails[25].

Quanto aos mecanismos de autenticação utilizados verificou-se que a sua diversidade é muito grande. Mesmo quando os serviços utilizam modelos parecidos, detalhes de implementação são diferentes. Ao pensar em fazer integração automática é praticamente impossível contemplar todos os mecanismos de autenticação existentes. Iniciativas de padronização de autenticação, como o padrão OAuth, devem ser consideradas prioritárias numa estratégia de automatização de integração, devido a possibilidade de, no

[24] http://code.google.com/apis/gdata/index.html
[25] http://www.rubyonrails.com/

futuro, um grande número de aplicações venham a aderir a esta solução. Segundo este critério, devem ser levadas em conta também algumas implementações proprietárias como Google AuthSub ou Yahoo BBAuth, devido ao grande número de aplicações que já as implementam.

## 7 Conclusões

A integração de aplicações Web 2.0 através de suas APIs é uma possibilidade relativamente recente para o desenvolvimento de aplicações na Web e não existem ainda *frameworks* teóricos que descrevam todas as suas dimensões. Este artigo apresenta uma rede sistêmica que organiza os principais aspectos relacionados à componentização das aplicações da Web 2.0, com o intuito de facilitar o estudo e a análise das APIs oferecidas por estas aplicações. Os resultados desta análise podem servir para apontar direções para o desenvolvimento de ferramentas que facilitem a integração de aplicações existentes, assim como auxiliar os desenvolvedores de novas APIs na criação de soluções mais facilmente integráveis.

A principal conclusão obtida a partir deste processo de classificação é a constatação de que uma grande parte das APIs pode ser considerada híbrida, implementando, simultaneamente, características das arquiteturas REST e RPC, o que dificulta a integração automática destas aplicações. Assim, uma alternativa neste sentido, é descrever formalmente as APIs através de linguagem de descrição como WSDL (*Web Services Description Language*) [2] ou WADL (*Web Application Description Language*) [10].

Outra conclusão foi a confirmação de que a diversidade de mecanismos de autenticação encontrada impossibilita o desenvolvimento de um procedimento único capaz de lidar com todos os processos de autenticação utilizados por estas APIs. Embora exista uma tendência de padronização através do OAuth, as muitas soluções proprietárias existentes precisam ser levadas em consideração.

Este trabalho está inserido no contexto de um estudo mais amplo onde foi definida uma proposta de integração de aplicações que resultou na implementação de um ambiente de aprendizagem baseado nestas idéias [3,4]. Neste ambiente é possível incorporar aplicações com API REST híbrida automaticamente, desde que as mesmas sejam descritas formalmente. Para isso, entretanto, foi necessário propor uma extensão ao WADL que permitisse especificar o modelo de autenticação utilizado.

## Referências


1. Berners-Lee, T., Fielding, R, Masinter, L: Uniform Resource Identifiers (URI): Generic Syntax, RFC 3986. ftp://ftp.isi.edu/in-notes/rfc3986.txt (2005)
2. Booth, D., Liu, C.K.: Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. W3C Recommendation, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/ (2007)
3. Bomfim, M.N.C.: Integração Automática de Aplicações Externas em um Ambiente de Aprendizagem Apoiado na Web 2.0. Dissertação (Mestrado em Informática) – Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro (2009)
4. Bomfim, M.N.C., Assis, J.S.S., Sampaio, F.F.: Avance: um ambiente de ensino e aprendizagem baseado na Web 2.0. In: Conferência Ibero-Americana WWW/Internet - CIAWI 2007. Vila Real – Portugal (2007).
5. Bliss, J., Monk, M, Ogborn, J.: Qualitative Data Analysis for Educational Research: A guide to uses of systemic networks. London: Croom Helm (1983)
6. Crockford, D.: JSON: The fat-free alternative to XML. In: Proc. of XML 2006, Boston, USA, http://www.json.org/fatfree.html (2006)
7. Fielding, R.T., Taylor, R.N.: Pricipled Design of the Modern Web Architecture. In: ACM Transactions on Internet Technology, Vol 2, No. 2, May 2002, pp 115-150 (2002)
8. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation, University of California, Irvine, USA, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (2000)
9. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E., Stewart, L: HTTP Authentication: Basic and Digest Access Authentication, RFC 2617, http://tools.ietf.org/html/rfc2617 (1999)

10. Hadley, M.: Web Application Description Language (WADL) Specification. Sun Microsystems Inc. https://wadl.dev.java.net/ (2006)
11. Mitra, N., Lafon, Y.: SOAP Version 1.2 Part 0: Primer (Second Edition). W3C Recommendation, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (2007)
12. Nelson, B.J.: Remote Procedure Call. Ph.D. dissertation CMU-CS-81-119, Carnegie-Mellon University (1981)
13. O'Reilly, T.: What is Web 2.0? Design Patterns and Business Models for the Next Generation of Software, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (2005)
14. Rescorla, E.: SSL and TLS: Designing and Building Secure Systems, Addison-Wesley (2001)
15. Richardson, L.; Ruby, S.: RESTful Web Services. First Edition. O'Reilly, 419 p. (2007)