Desenvolvimento de Aplicações Sociais a partir de APIs em Redes Sociais *Online*

Otávio C. Xavier *Cedric L. de Carvalho*

Technical Report - RT-INF_001-11 - Relatório Técnico
February - 2011 - Fevereiro

Instituto de Informática
Universidade Federal de Goiás
www.inf.ufg.br

Desenvolvimento de Aplicações Sociais A Partir de APIs em Redes Sociais *Online*

Otávio C. Xavier [*] otaviocx@gmail.com

Cedric L. de Carvalho [†] cedric@inf.ufg.br

Resumo. *Redes Sociais Online são sistemas Web para relacionamento e troca de conhecimentos entre pessoas. Um dos frutos da Web 2.0, as redes sociais podem ser de propósito geral ou para um público específico (músicos, médicos, pessoas solteiras, etc.). Com a popularidade, as redes sociais online foram ganhando mais funcionalidades, tornando a interatividade entre os usuários cada vez maior. Atualmente, muitas redes sociais online possuem APIs para que os próprios usuários possam criar aplicações para elas. Este trabalho destina-se ao estudo das APIs das redes sociais mais populares (como Facebook, Twitter, Orkut, Linkedin, Flickr e MySpaces). Tais APIs serão descritas com exemplos de implementações. Também serão vistas tecnologias como OpenSocial, OAuth e OpenID, que auxiliam no desenvolvimento de aplicações para redes sociais.*

Palavras-Chave: Redes Sociais *Online*, API, Facebook, Twitter, Flickr, OpenSocial, OpenID, OAuth.

1 Introdução

Desde sua origem, as redes sociais *online* (também chamadas de sites de rede social) têm atraído milhões de usuários em todo o mundo. Muitos destes integraram tais sites em suas rotinas diárias. Atualmente existem centenas de redes sociais *online*, com várias implementações tecnológicas diferentes, dando apoio à uma grande quantidade de práticas e interesses. Algumas redes sociais caracterizam-se por serem um canal de comunicação de sites pré-existentes. Outras ajudam estranhos a se conhecerem, baseadas nos interesses em comum das pessoas. Existem redes sociais *online* de âmbito geral, mas também aquelas destinadas à pessoas de um país, raça, língua, religião ou interesses específicos [3].

Os sites de rede social são baseados nos relacionamentos reais entre pessoas. Com isso, muitos destes sites destinam-se à formação de redes *online* entre pessoas que já se conhecem pessoalmente. A rede de cada usuário conecta-se com redes de outros usuários formando uma grande rede em cada um dos sites de rede social. Essas grandes redes podem ser usadas como plataformas para aplicações sociais. Tais aplicações necessitam do colaborativismo e do relacionamento entre as pessoas, presentes em redes sociais. Visando explorar essa possibilidade,

[*] Mestrando em Ciência da Computação, INF-UFG

[†] Orientador

alguns sites de rede social desenvolveram APIs[1], permitindo que os próprios usuários construam suas próprias aplicações sociais.

Este trabalho destina-se a descrever, testar e implementar aplicações através das APIs de redes sociais *online* mais comuns (como o Facebook [15], Twitter [23], LinkedIn [29], Flickr [28] e OpenSocial [5]). Também será visto o OpenID [31], um arcabouço para autenticação única de usuários em várias aplicações e o OAuth [19], um protocolo para autorização segura de aplicações para acesso a dados de outras aplicações. A principal motivação para tal estudo está na possibilidade de desenvolvimento de aplicações sociais que interconectem os usuários de redes sociais *online* distintas. Outro fator motivante para este trabalho está associado à Web Semântica. Atualmente, são poucos os sites de rede social que tratam semanticamente as informações e relacionamentos entre seus membros.

2 Redes Sociais *Online*

Sites de redes sociais podem ser definidos como serviços, baseados na web, que permitem aos seus usuários [3]:

1. Construir um perfil público ou semi-público em um sistema delimitado;
2. Elaborar uma lista de amigos com quem eles compartilham uma conexão e
3. Ver e navegar pela sua lista de conexões e pelas de outras pessoas.

A natureza e a nomenclatura dessas conexões podem variar de acordo com o site. Apesar de os sites de rede social atuais implementarem uma série de funcionalidades, sua espinha dorsal consiste em perfil público para cada usuário que mostra seus dados e a lista de conexões dele [3]. Após cadastrar-se em um site de rede social, é exibido um formulário ao usuário, com perguntas sobre ele. O perfil é formado a partir das respostas do usuário que podem incluir idade, localização, interesses, uma foto e um texto auto-descritivo do usuário.

Depois que o usuário cria seu perfil, o próximo passo em um site de rede social é identificar os usuários já cadastrados que ele deseja ter como conexões. As nomenclaturas mais comuns para as conexões são "Amigos", "Contatos", "Fãs"ou "Seguidores". Tais conexões podem ser bidirecionais ou unidirecionais. Conexões bidirecionais requerem que os dois indivíduos a aceitem, já as unidirecionais não. As nomenclaturas mais comuns para as conexões unidirecionais são "Seguidores"e "Fãs". É comum os sites de rede social possuírem também álbum de fotos e uma área em que os usuários podem se comunicar através de mensagens de texto [3].

Como os sites de rede social incentivam os usuários a colocarem dados pessoais e íntimos, pessoas mal intencionadas podem usar tais dados para violar a segurança dos usuários. Para aliviar esse problema, alguns sites de rede social adotaram níveis de privacidade. Os usuários têm a opção de escolher, por exemplo, que apenas seus contatos vejam seus dados pessoais.

2.1 Histórico

De acordo com a definição acima, o primeiro site de rede social surgiu em 1997. O SixDegrees.com permitia que seus usuários cadastrassem perfis e montassem uma lista de amigos, com perfis de outros usuários. A partir de 1998, os usuários do SixDegrees.com podiam

[1] *Application Programming Interfaces* descrevem interfaces com um conjunto de funcionalidades, visando o seu reuso.

navegar nas listas de amigos de outros usuários. Muitos sites já implementavam alguma dessas funcionalidades antes do SixDegrees.com. Entretanto, ele foi o primeiro a combiná-las [3]. O SixDegrees.com promovia-se como sendo uma ferramenta para ajudar pessoas a se conectarem e enviar mensagens para outras. Enquanto ele atraia milhões de usuários, começou a tornar-se um negócio inviável. Em 2000 o serviço foi fechado [3].

De 1997 a 2001, várias ferramentas de comunidade que combinavam perfis e listas públicas de amigos foram lançadas. Uma delas foi o LiveJournal [26], lançado em 1999. Ele permite que os usuários criem "jornais pessoais". Cada pessoa pode montar uma página com as notícias de seu interesse. O LiveJournal permite conexões unidirecionais. Um usuário recebe as notícias dos usuários em sua lista de conexões. Outros sites, implementaram características de rede social após seu surgimento. A exemplo o coreano Cyworld, fundado em 1999 e que implementou funcionalidades de redes sociais em 2001 e o sueco LunarStorm, que se tornou um site de rede social em 2000 [3].

A próxima geração de sites de rede social veio com o Ryze.com, o Fotolog e o Friendster. Ryze.com, criado em 2001, ajudava as pessoas a montar redes sociais de negócios. O Fotolog surgiu em 2002, com uma ideia semelhante a de um blog, entretanto com fotos como tema central de cada artigo. O Friendster, criado em 2002, surgiu como um complemento ao Ryze. Enquanto todos os sites sociais da época focavam em encontros amorosos entre estranhos, o Friendster focava em amigos de amigos (*friends-of-friends*). Seus criadores Jonathan Abrams e Cris Emmanuel assumiram que encontros amorosos em que o casal possuia um amigo em comum, eram mais bem sucedidos [3]. Entretanto, os usuários do Friendster também o usavam para aumentar as suas redes de amigos mais rápido que com encontros presenciais [33]. Com seu sucesso, em apenas um ano o Friendster já possuía mais de 30 milhões de usuários [3]. Contudo, tamanho sucesso acabou acarretando dificuldades técnicas, gerando uma evasão de usuários.

A partir de 2003, redes sociais tornaram-se comuns em sites da Web, como pode ser visto na Figura 1. Sites como LinkedIn, Visible Path e Xing formam redes sociais profissionais. No LinkedIn, por exemplo, os usuários podem cadastrar em seu perfil um currículo acadêmico e profissional. Tal currículo pode ser usado por empresas na seleção de profissionais. No Couchsurfing os usuários compartilham hospedagens. Um usuário pode oferecer hospedagem a outro usuário que deseja viajar para a sua cidade. O MyChurch conecta igrejas cristãs e seus membros. Alguns sites usam os conceitos de redes sociais *online* para compartilhamento de conteúdo multimídia. A exemplo, o Flickr para compartilhamento de fotos, o YouTube para vídeos e o Last.FM para músicas [3].

O MySpace, criado em 2003, cresceu rapidamente, como sendo uma alternativa ao então saturado Friendster. Apesar de ter sido criado com propósito similar ao do Friendster, o MySpace ganhou popularidade entre bandas de rock. Até 2004, a maioria dos seus usuários eram músicos. Em 2005, ele tornou-se o maior site de rede social (em número de usuários e visitantes) do mundo [3]. O Orkut, criado em 2004, é a rede social do Google. Desde sua criação, ganhou popularidade entre os brasileiros, tornando-se a rede social mais usada no Brasil [14]. O Facebook, criado em 2004, possuía um foco diferente de seus precursores. Ele foi criado para a formação de redes em faculdades específicas. Inicialmente, estava disponível apenas para os estudantes de Harvard. Em 2005, ele se expandiu para escolas de ensino médio, posteriormente para profissionais e redes corporativas e atualmente para qualquer pessoa. O principal diferencial do Facebook desde sua criação, era a API que possibilitava aos usuários desenvolver suas próprias aplicações embutidas à ele [3].

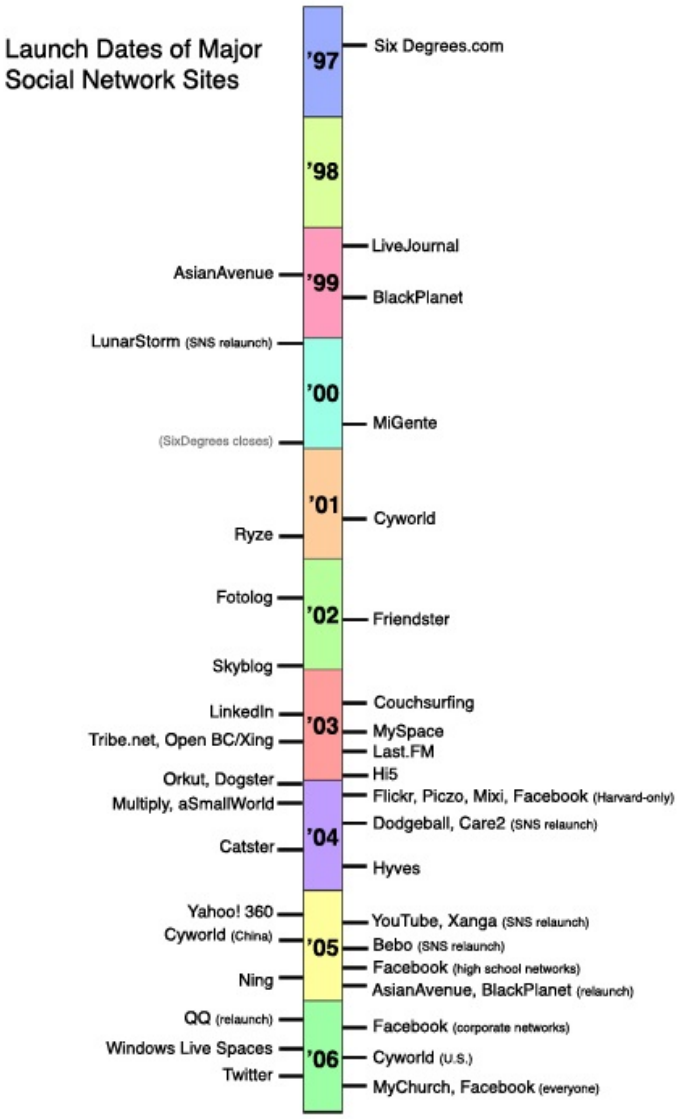

Figura 1: Linha do tempo do surgimento dos principais sites de rede social até 2006. [3]

Em 2006, o Twitter foi criado, gerando um novo conceito: o *microblogging*. Os usuários escrevem pequenas mensagens (de até 140 caracteres), sobre algo que está sentindo ou acontecendo. As conexões são unidirecionais. Apenas os usuários que estão conectados a uma pessoa, recebem as mensagens dela [24]. Com esse novo conceito, o Twitter cresceu rapidamente, tornando-se um novo meio de comunicação. Atualmente ele é usado para diversos fins como publicação de notícias, sorteios, venda de produtos e reuniões entre outros. Como o Facebook, o Twitter foi criado com uma vasta API para criação de aplicações pelos próprios usuários. A partir do sucesso das APIs do Facebook e do Twitter, em 2007 o Google lançou o OpenSocial [22]. Um conjunto de APIs para criação de aplicações Web em sites de redes sociais. Sites como Orkut, MySpace e LinkedIn são compatíveis com o OpenSocial para criação de aplicações. Para facilitar a integração ao OpenSocial, a Apache desenvolveu o Shindig [13], um *container* open-source para hospedagem de aplicativos em OpenSocial.

3 APIs Para Desenvolvimento de Aplicações em Sites de Redes Sociais

Como toda aplicação Web, sites de redes sociais são hospedados em servidores Web. Para que outras aplicações consigam interagir com aplicações Web, são usados *Web Services*. Então, as APIs de redes sociais *online* são *Web Services* destinados a oferecer recursos e funcionalidades, presentes na rede social, para aplicações externas. Essas APIs transformam os sites de rede social em plataformas para aplicações sociais. Nessa seção serão descritas e exemplificadas as APIs das principais redes sociais *online*.

As APIs Web de Redes Sociais *Online* são *Web Services* que permitem acesso e manipulação de informações sociais. A maioria das APIs são implementadas através dos princípios REST[9]. REST, ou Transferência de Estado Representacional, termo introduzido por Roy Fielding em sua tese de doutorado, descreve um estilo de arquitetura para sistemas hipermídia derivado de vários estilos baseados em redes como a Web [9]. Nessa técnica as mensagens trocadas são encapsuladas diretamente no protocolo HTTP. Há um foco nos recursos e não nas chamadas aos procedimentos/serviços, como em outras abordagens. No REST são feitas requisições para recursos e não serviços. Essa abordagem pode ser interessante para aplicações em que é mais importante a interoperabilidade do que um contrato formal entre as partes [34]. Os recursos são representados por URIs e também podem ser chamados de métodos, caso representem alguma ação.

3.1 OAuth

OAuth ou *Open Authorization* é um padrão aberto que permite aos usuários de um site garantirem acesso, por uma aplicação externa, aos seus recursos privados sem ter que compartilhar suas senhas e logins [19]. O foco principal do OAuth está na autorização e não na autenticação. Sendo assim, ele prevê níveis de autorização, que o usuário pode aceitar ou não. O OAuth foi construído baseado nos padrões proprietários Google AuthSub, Yahoo BBAuth e Flickr API Auth. Também pode ser caracterizado como um protocolo. Sua especificação consiste em duas partes. A primeira parte define um processo no navegador, para redirecionamento do usuário final para autorização aos seus recursos, pelo cliente[2]. A segunda parte define um método para realização de requisições HTTP autenticadas usando dois conjuntos de credenciais. Um conjunto destinado à identificação do cliente e outro à identificação do dono do recurso a ser requisitado [17]. O protocolo segue o seguinte fluxo:

1. ***Token* de Requisição**. Quando o usuário entra na aplicação consumidora, esta requisita ao servidor um *token* de requisição. Quando a aplicação consumidora recebe o *token*, ela redireciona o usuário para a tela de autenticação do servidor. O *token* de requisição é enviado, bem como um link para redirecionamento, assim que o usuário autenticar-se.

2. **Autenticação e Autorização**. Ao ser redirecionado para a tela de autenticação do servidor, o usuário é requisitado a autenticar-se. Uma vez autenticado, o usuário recebe um questionamento acerca da autorização para a aplicação consumidora.

3. **Redirecionamento à aplicação consumidora**. Caso o usuário autorize o acesso, o servidor marca o *token* de requisição, enviado no passo 1, como autorizado. O usuário então é redirecionado para o URI previamente informado pela aplicação consumidora.

[2]No OAuth, o cliente ou consumidor é a aplicação que está requisitando os recursos privados. O site que os fornece é chamado de servidor ou provedor do serviço. O usuário é chamado de dono do recurso.

4. ***Token* de acesso**. Quando o fluxo retorna à aplicação consumidora, ela encarrega-se de fazer um intercâmbio do *token* de requisição por um *token* de acesso. O *token* de requisição tem como único objetivo a aprovação, pelo usuário. Entretanto, o *token* de acesso é destinado às requisições dos recursos privados.

5. **Acesso aos recursos privados**. Com o *token* de acesso, a aplicação consumidora pode consultar todos os recursos privados, aos quais o usuário concedeu permissão.

O OAuth foi desenvolvido em 2006. Com mais de 3 anos de experiência trabalhando com esse protocolo, em 2009 a comunidade começou a repensá-lo. A versão 2.0 do OAuth [20] é um protocolo completamente novo. Sendo assim, ela não garante compatibilidade com as versões anteriores. Essa versão ganhou melhorias em 3 grandes áreas em que a versão 1.0 era limitada ou confusa [18]:

1. **Autenticação e Assinaturas**. A principal falha das tentativas de implementação do OAuth 1.0 foi causada pela complexidade da criptografia requerida na especificação. Mesmo depois de uma revisão, para a versão 1.0a, OAuth continuou não trivial para uso no lado do cliente[3]. Para usar a especificação, os desenvolvedores eram obrigados a buscar, instalar e configurar bibliotecas externas.

2. **Experiência do Usuário e Opções Alternativas de Emissão de *Tokens***. OAuth inclui duas partes principais: obtenção do *token*, através da autorização de acesso pelo usuário e uso do *token* para obtenção de recursos protegidos. O método para obtenção do *token* é chamado *flow*. A versão 1.0 do OAuth possuia 3 *flows* (*web, desktop e mobile*). Entretanto, a especificação evoluiu posteriormente para um único *flow*, que atendia os três tipos de clientes. Porém, com o aumento do número de sites que usam o OAuth, o *flow* disponível acabou se tornando cada vez mais limitado.

3. **Performance em Escala**. Com grandes empresas aderindo ao OAuth, a comunidade chegou à conclusão de que o protocolo não possuía um bom escalonamento. Ele necessitava de gerenciamento de estados através de diferentes etapas e requeria a emissão de credenciais de longa duração, que são menos seguras e mais difíceis de gerir.

Com a versão 2.0 do OAuth foram criados 6 novos *flows*:

- ***User-Agent Flow***. Para clientes em navegadores Web.
- ***WebServer Flow***. Para clientes que são parte de uma aplicação em servidor Web.
- ***Device Flow***. Adequados para clientes de execução em dispositivos limitados.
- ***Username and Password Flow***. Usado quando há um elevado grau de confiança entre o usuário e o cliente, para guardar suas credenciais.
- ***Client Credentials Flow***. O Cliente usa suas credenciais para obter o *token* de acesso.
- ***Assertion Flow***. O cliente apresenta uma afirmação como uma SAML[4] para o servidor de autorização, em troca de um *token* de acesso.

[3]Lado do cliente, nesse contexto, refere-se às linguagens interpretadas pelo navegador, como JavaScript e HTML.

[4]Security Assertion Markup Language. Mais detalhes em `http://saml.xml.org/`

A versão 2.0 do OAuth ainda provê uma opção para autenticação sem criptografia, utilizando HTTPS. As assinaturas foram simplificadas, não sendo mais necessária a análise, codificação e a ordenação de parâmetros. Nesta versão, é necessário apenas uma chave secreta. Também são implementados *tokens* de acesso de curta duração, que são renovados a partir de *tokens* de atualização. Isso permite ao cliente requisitar um novo *token* de acesso, de forma transparente ao usuário. O OAuth 2.0 separa o papel de autenticação e autorização pelo servidor, da requisição de recursos através da API. Isso facilita o uso de servidores distribuídos [18].

3.2 API do Facebook

O Facebook possui uma API para leitura e escrita de dados em seu núcleo. Ela é chamada de *Graph API*. Com ela é possível, de maneira simples, requisitar informações como o relacionamento entre usuários, suas fotos, gostos, eventos, páginas entre outras coisas. Para que uma aplicação consiga acessar as funcionalidades da *Graph API* é necessário passar por um processo de autorização. A aplicação só poderá consultar informações de um usuário que a autorizou. Para isso é usado o protocolo OAuth (mais detalhes na seção 3.1). A plataforma do Facebook usa a versão 2.0 do OAuth, seguindo o seguinte fluxo:

1. **Registrar uma aplicação**[5]. Cada aplicação registrada possui um ID e um código secreto, necessários para autenticação.

2. **Redirecionar o usuário para o processo de autorização**. Para isso é necessário informar o ID da aplicação e um URI para redirecionamento após a etapa de autorização, como no exemplo que pode ser visto em Código 1.

Código 1 – URI de autorização da API do Facebook.

```
https://graph.facebook.com/oauth/authorize?
client_id=...&
redirect_uri=http://www.example.com/oauth_redirect
```

3. **Código de segurança**. Se o usuário autorizar a aplicação, ele será redirecionado para o URI informado no segundo passo. Será enviado para esse URI um parâmetro `code`. Tal parâmetro será usado para requisitar um *token* de acesso.

4. ***Token* de acesso**. O código de segurança deve ser trocado por um *token* de acesso. Esse *token* deverá ser enviado para a API em toda requisição que necessite autorização. Para requisitá-lo é necessário enviar o ID da aplicação, seu código secreto, o código de segurança e o URI para redirecionamento. Como no exemplo exibido em Código 2.

Código 2 – Requisitando Token de acesso, para aplicações, da API do Facebook.

```
https://graph.facebook.com/oauth/access_token?
client_id=...&
redirect_uri=http://www.example.com/oauth_redirect&
client_secret=...&
code=...
```

[5]Link para registro de aplicações no Facebook: http://developers.facebook.com/setup/

Se, no passo 3, o usuário não autorizar a aplicação, o mesmo URI de redirecionamento será chamado. Entretanto, será enviado o parametro `error_reason`.

A autenticação, conforme descrita acima, garante acesso às informações públicas do usuário. As informações públicas são aquelas que estão acessíveis em seu perfil, para qualquer outro usuário. Elas incluem o nome, a foto do perfil, a lista de amigos entre outras coisas. Entretanto, a aplicação pode necessitar de maior permissão. Para isso o Facebook possui alguns grupos de permissões estendidas. Acesso ao álbum de fotos do usuário, gerenciamento de eventos e páginas, postar no mural de recados e acesso ao chat são exemplos de funções privadas e requerem permissões estendidas. Para requisitá-las, um parâmetro extra (`scope`) deve ser enviado no processo de autorização. O exemplo presente em Código 3 substitui o URI a ser chamado no exemplo do passo 2.

Código 3 – Requisitando acesso com permissões estendidas, na API do Facebook.

```
https://graph.facebook.com/oauth/authorize?
client_id=...&
redirect_uri=http://www.example.com/callback&
scope=user_photos,user_videos,publish_stream
```

O exemplo em Código 3 requisitará ao usuário 3 permissões estendidas para a aplicação. Acessar suas fotos, vídeos e publicar em seu mural, respectivamente. Uma lista completa das permissões estendidas pode ser obtida em [8]. A tela de login e a requisição de permissões estendidas são feitas a partir de uma janela *pop-up*. Entretanto, para maior interoperabilidade, pode ser enviado o parâmetro `display` pelo URI. Esse parâmetro pode assumir os valores `page`, para autenticação em uma nova página, `popup`, para janela em *pop-up*, `wap` para uma versão *WAP*, otimizada para dispositivos móveis e `touth` para *smartphones* como *Android* e *iPhone*.

Para obter dados acerca de aplicações, não é preciso autorização de usuário. Para isso, há um procedimento mais simplificado. O *token* de acesso pode ser requisitado como no exemplo exibido no Código 4.

Código 4 – Requisitando Token de acesso, para aplicações, da API do Facebook.

```
https://graph.facebook.com/oauth/access_token?
client_id=...&
client_secret=...&
type=client_cred
```

O diferencial é o parâmetro `type`. O *token* retornado poderá ser usado como no exemplo presente no código 5.

Código 5 – Usando Token de acesso da API do Facebook.

```
https://graph.facebook.com/app_id/insights?access_token=...
```

No exemplo acima, `app_id` é o ID da aplicação. Esse exemplo requisita dados estatísticos acerca do uso da aplicação.

Todos os retornos da API são feitos no formato JSON[1]. Um formato para representação textual de dados de maneira leve e simplificada, baseado na sintaxe do JavaScript[1]. Para facilitar o uso da API, o Facebook dispõe de alguns Kits de Desenvolvimento (SDKs). Existe kits

para as linguagens de programação JavaScript, PHP e Python e para os sistemas operacionais, de *smartphones*, iOS e Android. No Código 6 é possível ver um exemplo de uso do *JavaScript SDK* para listar os nomes de amigos do usuário logado. O código é embutido em um HTML e é internacionalizável. Para mudar o idioma, basta trocar `pt_BR` na linha 1 pelo idioma a ser escolhido.

Código 6 – Requisitando Lista de Amigos com JavaScript SDK

```
0
```

A API do Facebook também possui uma linguagem para consulta de dados. Parecida com o SQL, ela é chamada de FQL (*Facebook Query Language*). O exemplo do Código 7 retorna o nome, a foto e o gênero de um usuário.

Código 7 – Exemplo de uso da FQL na API do Facebook.

```
https://api.facebook.com/method/fql.query?
query=SELECT name, pic, sex FROM user WHERE uid = 1820391700
```

A FQL pode retornar os dados no formato XML ou JSON. O padrão é XML. Para especificar o formato, basta enviar o parâmetro `format` com valor igual a `xml` ou `json`. Uma lista completa de tabelas e campos pode ser obtida em [7].

Também há uma linguagem de marcação, chamada de FBML (*Facebook Markup Language*). Para usá-la é necessário criar uma página interna ao Facebook ou usar o JavaScript SDK. Na linha 7 do Código 6, é ativado o interpretador de FBML. Uma lista completa com todas as tags da FBML pode ser consultada em [6].

3.3 API do Twitter

A API do Twitter consiste em três partes. Duas delas são APIs Web baseadas no REST e uma para fluxo contínuo de dados (*Streaming*). Como dito anteriormente na Seção 2.1, no Twitter cada usuário possui um histórico de pequenas mensagens sobre coisas que estão acontecendo ou que ele está sentindo. As duas APIs baseadas em REST, estão inteiramente relacionadas com o histórico. Há uma API para buscas (*Search API*) e outra para manipulação dos dados. A API de fluxo contínuo (*Streaming API*), possui arquitetura diferente das demais, com conexões de longa duração [10] [27] [30].

Para autenticação e autorização, a API do Twitter também utiliza o protocolo OAuth. Entretanto, na versão 1.0a. Essa versão é mais "burocrática"que a versão 2.0 (como visto na seção 3.1), sendo necessário o envio de informações no cabeçalho HTTP de requisição. A API do Twitter segue o seguinte fluxo [30]:

1. **Registrar uma aplicação**[6]. Cada aplicação registrada possui uma chave de API (*API Key*), uma chave de consumidor (*Consumer Key*) e um código secreto de consumidor (*Consumer Secret*), necessários para autenticação.

2. ***Token* de requisição**. Esse *token* é necessário para a autenticação. Ele será trocado pelo *token* de acesso posteriormente. Para requisitá-lo é necessário passar alguns parâmetros:

[6]Link para registro de aplicações no Twitter: http://dev.twitter.com/apps

- `oauth_nonce`: Uma chave aleatória usada para prevenir ataques.
- `oauth_callback`: Um URI para redirecionamento após a obtenção do *token* de requisição. No exemplo abaixo foi utilizado o URI `http://localhost/process_callback`.
- `oauth_consumer_key`: A chave de consumidor, adquirida no cadastro da aplicação.
- `oauth_version`: A versão do OAuth a ser usado (2.0 ainda não está implementado).
- `oauth_signature_method`: O método de criptografia para a assinatura. Atualmente o Twitter suporta apenas HMAC-SHA1[7]
- `oauth_timestamp`: O timestamp atual, em formato Unix.
- `oauth_signature`: Uma assinatura gerada a partir da concatenação do método de requisição com o URI requisitado e os parâmetros passados, em ordem alfabética. Os parâmetros devem ser colocados no padrão *URL-enconde*, como no exemplo presente no Código 8.

Código 8 – Exemplo de string para geração do parâmetro `oauth_signature`.

```
POST&
https\%3A\%2F\%2Fapi.twitter.com\%2Foauth\%2Frequest_token&
oauth_callback\%3Dhttp\%253A\%252F\%252Flocalhost\%252F
                    process_callback\%26
oauth_consumer_key\%3DGDdmIQH6jhtmLUypg82g\%26
oauth_nonce\%3DQP70eNmVz8jvdPevU3oJD2AfF7R7odC2XJcn4XlZJqk\%26
oauth_signature_method\%3DHMAC-SHA1\%26
oauth_timestamp\%3D1272323042\%26
oauth_version\%3D1.0
```

A assinatura consiste no *hash* gerado pela concatenação acima, a partir do algoritmo informado no parâmetro `oauth_signature_method`. O algorítimo usado pelo Twitter necessita de chave. Com isso, é usada a *Consumer Secret* da aplicação, nesse momento, para geração da assinatura.

O *token* de requisição é obtido através do URI `https://api.twitter.com/oauth/request_token`. Os parâmetros são passados pelo cabeçalho *Authorization* do protocolo HTTP, como no exemplo exibido no Código 9.

[7]HMAC é uma sigla para *Hash-based Message Authentication Code*. HMAC-SHA1 é uma criptografia sem volta que usa o algoritmo SHA1 e uma chave.

Código 9 – Cabeçalho *Authorization* do protocolo HTTP, passado pelo OAuth.

```
OAuth
oauth_nonce="QP70eNmVz8jvdPevU3oJD2AfF7R7odC2XJcn4XlZJqk",
oauth_callback="http\%3A\%2F\%2Flocalhost\%2Fprocess_callback",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="1272323042",
oauth_consumer_key="GDdmIQH6jhtmLUypg82g",
oauth_signature="8wUi7m5HFQy76nowoCThusfgB\%2BQ\%3D",
oauth_version="1.0"
```

O retorno dessa requisição é uma *string* com 3 parâmetros concatenados. `oauth_token`, o token de requisição; `oauth_token_secret`, uma chave secreta que será usada no passo 4 e `oauth_callback_confirmed`, que informa se o URI de retorno foi entendido corretamente.

3. **Autorizando a aplicação pelo usuário**. O *token* obtido no passo anterior deverá ser enviado para o URI de autenticação/autorização. O URI pode ser `https://api.twitter.com/oauth/authorize` ou `https://api.twitter.com/oauth/authenticate`. A diferença está no fato de que com o método `oauth/authorize` o usuário terá que confirmar a autorização em todas as requisições, enquanto que com o método `oauth/authenticate` o usuário só precisa autorizar a aplicação uma vez. Caso não haja usuário logado, será exibido um formulário para *login* de usuário. Após sua autenticação, o usuário é requisitado a autorizar a aplicação. Uma vez autorizado o acesso, o fluxo é redirecionado para o URI de retorno informado no passo 2. Nesse redirecionamento são enviados dois parâmetros: `oauth_token` e `oauth_verifier`, que serão usados no próximo passo.

4. ***Token* de acesso**. Com a autorização pelo usuário, o último passo é obter um *token* que será usado em todas as requisições futuras. Para isso, é feita uma requisição semelhante ao do passo 2, para o URI `https://api.twitter.com/oauth/access\_token`, com os seguintes parâmetros:

   - `oauth_nonce`: Uma chave aleatória usada para prevenir ataques.
   - `oauth_consumer_key`: A chave de consumidor, adquirida no cadastro da aplicação.
   - `oauth_version`: A versão do OAuth a ser usado (2.0 ainda não está implementado).
   - `oauth_signature_method`: O método de criptografia para a assinatura. Atualmente o Twitter suporta apenas HMAC-SHA1[8]
   - `oauth_timestamp`: O timestamp atual, em formato Unix.
   - `oauth_token`: O *token* de requisição, reenviado para o URI de retorno no passo anterior.
   - `oauth_verifier`: Um código verificador, gerado pela autorização do usuário. Esse código é enviado ao URI de retorno, no passo anterior.

[8]HMAC é uma sigla para *Hash-based Message Authentication Code*. HMAC-SHA1 é uma criptografia sem volta que usa o algorítimo SHA1 e uma chave.

- `oauth_signature`: Uma assinatura gerada a partir da concatenação do método de requisição com o URI requisitado e os parâmetros passados, em ordem alfabética. Os parâmetros devem ser colocados no padrão *URL-enconde*, como no exemplo exibido no Código 10.

Código 10 – Exemplo de string para geração do parâmetro `oauth_signature` para ***Token* de acesso**.

```
POST&
https\%3A\%2F\%2Fapi.twitter.com\%2Foauth%2Faccess_token&
oauth_consumer_key\%3DGDdmIQH6jhtmLUypg82g\%26
oauth_nonce\%3D9zWH6qe0qG7Lc1telCn7FhUbLyVdjEaL3MO5uHxn8\%26
oauth_signature_method\%3DHMAC-SHA1\%26
oauth_timestamp\%3D1272323047\%26
oauth_token\%3D8ldIZyxQeVrFZXFOZH5tAwj6vzJYuLQpl0WUEYtWc\%26
oauth_verifier\%3DpDNg57prOHapMbhv25RNf75lVRd6JDsni1AJJIDYoTY\%26
oauth_version\%3D1.0
```

Novamente, é gerado um *hash* da concatenação e ele é a assinatura. Entretanto, agora a chave para a criptografia será a concatenação entre o *Consumer Secret* e o `oauth_token_secret`, obtido no passo 2. O retorno da requisição é composto pelo *token* de acesso, uma palavra-chave secreta, o ID e o nome do usuário que autorizou a aplicação.

Pode-se observar que as requisições realizadas no Twitter, para autenticação e autorização, não são comuns. Isso se dá pelo uso da versão 1.0 do protocolo OAuth. Para fazer requisições como essas, é necessário um cliente em que seja possível a edição do cabeçalho HTTP. O cURL[9] é uma biblioteca interessante para essa funcionalidade. Os códigos 11 e 12 utilizam essa biblioteca em conjunto com a linguagem de programação PHP, como exemplo.

Código 11 – Script PHP para obtenção de tokens usando a função `sendOAuthReq` (código 12)

```
0
```

Código 12 – Função PHP para envio de requisições OAuth utilizando cURL

```
0
```

A API do Twitter ainda possui uma outra forma de autorização, que não usa OAuth. Essa forma é bem simplificada, chamada de *Basic Authentication*. Consiste no envio do usuário e senha pelo cabeçalho HTTP. Essa autenticação é insegura e está em desuso. Por isso, não será descrita neste estudo [27].

A API de busca (*Search API*) não necessita de autenticação pelo usuário. Ela pode ser usada, mesmo que não haja alguma aplicação registrada e vinculada às requisições. Essa API possui apenas o método `search`. Ela está separada do restante da API, por ter sido desenvolvida inicialmente fora do Twitter, por uma empresa chamada Summize. Entretanto, em 2008 o Twitter comprou tal empresa. Existem planos para unificação das duas APIs REST [30].

[9] Mais detalhes em `http://curl.haxx.se/`

O método `search` faz buscas e retorna-as em 4 possíveis formatos: xml, json, rss e atom. Ele pode ser requisitado a partir do URI: `http://search.twitter.com/search.format`, sendo que `format` pode assumir um dos 4 formatos disponíveis. A expressão para busca é enviada através do parâmetro `q`. O mecanismo de busca da *Search API* é otimizado e mistura resultados populares com resultados recentes. Entretanto, isso pode ser alterado a partir do parâmetro `result_type`. Ele pode ser definido como `mixed` (padrão), `popular` ou `recent`. Também há o parâmetro `geocode` para busca de twitter a partir da localização geográfica do usuário, entre outros.

A API principal do Twitter é responsável pela manipulação e consulta dos dados. Qualquer método dessa API é precedido do URI `http://api.twitter.com/version/`, sendo que `version` é a versão da API (atualmente 1). Nessa API, alguns métodos não requerem autorização/autenticação. Os métodos que trazem informações disponíveis no site do Twitter para qualquer visitante (não logado), não requerem autorização. Para esses métodos, é suficiente fazer uma requisição HTTP convencional, para seu URI. Um exemplo é o método `statuses/user_timeline`. Tal método retorna os últimos envios de um usuário em ordem cronológica. É possível especificar qual o usuário através dos parâmetros `user_id`, enviando o ID ou `screen_name`, enviando o *login* do usuário.

O método `trends` também não requer autorização. Ele retorna palavras que são tendências (muito faladas) no Twitter. Esse método possui os sub-recursos: `trends/current`, que retorna os termos mais falados atualmente; `trends/daily`, que mostra as expressões mais populares em um dia específico e `trends/weekly` para uma semana específica. Também há filtros por localidade, a partir do recurso `trends/available`, que pode receber latitude (`lat`) e longitude (`long`) como parâmetros.

Os métodos que requerem autenticação devem ser requisitados de maneira semelhante às requisições para obtenção do *token* de acesso. Os parâmetros comuns em qualquer requisição são `oauth_consumer_key`, `oauth_nonce`, `oauth_timestamp`, `oauth_version` e `oauth_signature_method`, já descritos. Também é necessário enviar o parâmetro `oauth_token`, com o *token* de acesso previamente obtido. Esses parâmetros são enviados pelo cabeçalho *Authorization*. Os parâmetros específicos do método requisitado são enviados por POST. A assinatura é formada conforme descrito anteriormente. Entretanto, também são concatenados os parâmetros passados por POST. A chave para a criptografia é composta pelo *Consumer Secret* e o `oauth_token_secret` obtido com o *token* de acesso. O Código 13 exibe um exemplo de requisição ao método `statuses/update`, usando a função `sendOAuthReq` do Código 12. Esse método insere um novo post na timeline do usuário autenticado[10].

Código 13 – Script PHP que chama método `statuses/update` usando a função `sendOAuthReq` (código 12)

```
0
```

A API de fluxo contínuo (*Streaming API*) possui métodos para busca e exibição dos *posts* de usuários. Todos os métodos dessa API requerem autorização, sendo que alguns requerem contato direto com os desenvolvedores da API para liberação. Os filtros são semelhantes aos presentes na API de busca. Entretanto, o fluxo é contínuo. Isso significa que, à medida que os usuários vão enviando *posts*, a API atualizada os dados para consulta, incluindo novos resulta-

[10]Uma lista completa com todos os métodos disponíveis na API do Twitter pode ser vista em: `http://dev.twitter.com/doc/`.

dos, sem a necessidade de novas requisições. O único formato permitido nessa API é o JSON[11] [27].

3.4 API do Flickr

Como adicional em relação à maioria das APIs de Redes Sociais *Online*, a API do Flickr suporta além da arquitetura REST, o SOAP e o XML-RPC. Entretanto, neste trabalho, será utilizada a arquitetura REST, em busca de maior semelhança e integração com as demais APIs. O Flickr trabalha com um protocolo de autenticação/autorização proprietário, semelhante ao OAuth. Tal protocolo foi um dos utilizados como inspiração para a elaboração do OAuth [18]. Como nas demais, na API do Flickr é necessário cadastrar uma aplicação[12]. Com o registro da aplicação, são criados uma chave e um segredo. Caso a aplicação seja Web, é necessário configurar um URL de retorno (*callback*). A autenticação segue o seguinte fluxo [21]:

1. **Código frob**. Inicialmente é feita uma requisição para o URI `http://flickr.com/services/auth/`. São enviados 3 parâmetros. O parâmetro `api_key` envia a chave da aplicação. O `api_sig`, deve ter como valor a assinatura desta requisição (gerada a partir do procedimento presente no passo 2). O `perms`, com o nível de permissões desejado para a aplicação. Esse parâmetro pode assumir 3 valores: `read` , para leitura de dados; `write` para inserção/atualização de dados e `delete` para exclusão de dados. Cada nível possui a permissão de seus antecessores. Logo, o nível `delete` possui `write` e `read`. Caso o usuário não esteja logado é exibido um formulário de *login*. Em seguida, o usuário é requisitado a aceitar ou negar autorização à aplicação. Caso o usuário aceite, a requisição redireciona o fluxo para o URL de retorno, enviando o parâmetro `frob`.

2. **Assinatura**. Para gerar a assinatura necessária no passo anterior, primeiramente ordena-se alfabeticamente os parâmetros a serem enviados. Em seguida, o segredo da aplicação é concatenado com os parâmetros e seus respectivos valores. A partir da expressão resultante, é calculado o *hash* MD5[13]. Esse *hash* será a assinatura.

3. **Obtendo *token* de acesso**. O aplicativo deverá então chamar o método `flickr.auth.getToken` da API. Os métodos da API do Flickr são requisitados a partir do URI `http://flickr.com/services/rest/`, para a arquitetura REST. O método desejado é informado pelo parâmetro `method`. Este método requer 3 parâmetros. A chave da aplicação, o código frob e uma assinatura, seguindo o mesmo procedimento do passo 2. A resposta terá um *token* para futuras chamadas de métodos.

Existem alguns métodos na API do Flickr que não necessitam de autorização. Entretanto, mesmo para estes é necessário enviar a chave da aplicação. O método `flickr.photos.search` é um exemplo. Ele tem como objetivo a busca por fotos em toda a base do Flickr. Este método possui uma série de parâmetros para filtros opcionais[14]. No Código 14 pode ser visto um exemplo de URI para requisitá-lo.

[11] Uma lista completa com todos os métodos disponíveis na API de fluxo contínuo do Twitter pode ser vista em: `http://dev.twitter.com/pages/streaming_api_methods`.

[12] O registro de aplicações para uso da API do Flickr pode ser feito em: `http://www.flickr.com/services/apps/`. É necessária uma conta de e-mail no Yahoo.

[13] MD5, *Message-Digest algorithm 5*, é um algoritmo de *hash* (criptografia sem volta) de 128 bits, desenvolvido pela RSA Data Security, Inc. e descrito na RFC 1321.

[14] A lista completa de parâmetros do método `flickr.photos.search` está disponível em [12]

Código 14 – Exemplo de URI, no Flickr, para o método `flickr.photos.search`.

```
http://www.flickr.com/services/rest/?
method=flickr.photos.search&
api_key=...&
tags=Goiania,Praça&
format=json
```

O URI acima retorna uma lista, em formato JSON, de dados de fotos que possuem as tags "Goiânia"e "Praça". A paginação é configurada pelos parâmetros `page` e `per_page`, com valores padrões de 1 e 100 respectivamente. O parâmetro `format` é opcional, sendo que o padrão é XML. Este parâmetro está presente em qualquer método da API. Parâmetros geográficos também estão disponíveis, como no exemplo exibido no Código 15:

Código 15 – Exemplo de URI, no Flickr, com parâmetros geográficos.

```
http://www.flickr.com/services/rest/?
    method=flickr.photos.search&
    api_key=...&
    text=paisagens&
    extras=geo,tags,description&
    lat=-16.71&lon=-49.20&radius=32
```

Neste exemplo é feita a busca por fotos que contenham a palavras "paisagens"no título, descrição ou tags e estejam geo-referenciadas nas coordenadas da cidade de Goiânia. Os parâmetros `lat` e `lon` refinam a busca em uma determinada localidade geográfica. O parâmetro `radius` informa o tamanho do raio em torno da coordenada informada (por padrão em km), em que será feita a busca. O parâmetro `extras` informa campos adicionais a serem retornados na busca. Para requisitar uma foto (e não apenas seus dados) é utilizada a sintaxe de URI exibida no Código 16:

Código 16 – Sintaxe de URI para requisição de fotos do Flickr.

```
http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}.jpg
```

Os valores entre chaves são parâmetros obtidos através do retorno do método de busca, conforme pode ser visto no Código 17.

Código 17 – Exemplo de tag do XML de retorno e URI para foto.

```
// Exemplo de tag do XML de retorno:
<photo id="1318815545" owner="10298760@N03"
       secret="0a1a634be8" server="1439" farm="2" ... />
// URI para a foto:
http://farm2.static.flickr.com/1439/1318815545_0a1a634be8.jpg
```

Existem métodos que referem-se aos dados do usuário autenticado. Para esses métodos é necessário conceder permissão `read`. Um exemplo é o método `flickr.contacts.getList`, que retorna uma lista de contatos do usuário logado. Há

o parâmetro opcional `filter` que filtra a busca por amigos, família, ambos ou outros. A seguir pode-se ver um exemplo de como utilizá-lo:

```
http://www.flickr.com/services/rest/?
    method=flickr.contacts.getList&
    api_key=...&
    auth_token=...&
    filter=family&
    api_sig=...
```

O URI acima retorna todos os contatos que estão marcados como família. É possível notar que são enviados a chave da aplicação, o *token* de acesso obtido na autenticação e a assinatura da requisição. Todos os métodos que requerem autorização, necessitam do envio desses 3 parâmetros [2].

Os métodos que atualizam dados ou interagem com outros usuários (como por exemplo, comentários, adicionar aos favoritos e adicionar notas) requerem permissão `write`. Um exemplo é o método `flickr.photos.setMeta`. Este método tem o objetivo de alterar o título e a descrição de uma foto. Devem ser enviados para esse método os parâmetros `photo_id`, com o id da foto; `title`, com o novo título da foto e `description`, com a nova descrição da foto. Métodos com permissão `write`, como esse, exigem solicitação HTTP POST.

Para exclusão de dados é necessária a permissão `delete`. Ao se autenticar e autorizar esse nível de permissão, o usuário concede à aplicação poder de exclusão de suas fotos e vídeos. Um exemplo de método que requer esse nivel de permissão é o `flickr.photos.delete`. Este método necessita do ID da foto que será excluída. É interessante observar que alguns métodos de exclusão como `flickr.favorites.remove`, para exclusão de favoritos e `flickr.photos.comments.deleteComment` para exclusão de comentários em uma foto, não necessitam de permissão `delete`. Métodos de exclusão que estão envolvidos apenas com a interação entre os usuários e não excluem recursos (fotos e vídeos), precisam apenas de permissão `write`. Uma lista completa dos métodos da API do Flickr pode ser obtida em [11].

3.5 OpenSocial

Diferentemente das APIs descritas anteriormente, o OpenSocial é um conjunto de APIs que possui como principal objetivo a interoperabilidade de aplicações em sites de rede social. Tais APIs descrevem um "modelo"de plataforma para desenvolvimento de aplicativos em redes sociais *online*. Os sites de redes sociais que implementam as APIs descritas pelo OpenSocial são chamados de recipientes. Os aplicativos desenvolvidos em OpenSocial funcionam em qualquer recipiente que suporte a versão em que este foi desenvolvido. Atualmente, mais de 40 sites de redes sociais são recipientes para o OpenSocial[15] [4]. Estão entre estes, sites como o Orkut, o MySpace, o LinkedIn, o Yahoo! e o Friendster. Entretanto, cada site implementa o OpenSocial em uma versão diferente. As versões mais atuais dão compatibilidade com as mais antigas, sendo que a atual é a 0.9. A API é completamente baseada em XML, HTML e JavaScript [4].

Com o OpenSocial, a aplicação fica encapsulada no site de rede social recipiente. Logo, questões de autenticação e autorização, para a aplicação, são definidas separadamente, no próprio recipiente. Entretanto, o OpenSocial possui suporte à OAuth, com foco na autorização para aplicações externas acessarem dados da aplicação desenvolvida em OpenSocial [16]. As aplicações desenvolvidas em OpenSocial consistem em um arquivo XML com chamadas

[15] A lista completa de recipientes para o OpenSocial pode ser vista em: http://wiki.opensocial.org/

à arquivos JavaScript e HTML, caso necessário. Tal arquivo XML centraliza as informações gerais sobre a aplicação. No cadastro de uma aplicação em OpenSocial, em um recipiente, deve ser informado o caminho para este arquivo XML.

O Código 18 exibe um exemplo simples de aplicação OpenSocial. Toda aplicação deve ter, como raiz, a tag Module. Em ModulePrefs são informadas as configurações e preferencias da aplicação, como ícone, imagem de exibição, nome, título, autor, língua padrão, entre outras coisas. O exemplo do Código 18 apenas exibe a mensagem "Olá Mundo!"em uma aplicação de nome "Teste - INF/UFG"e descrição "Teste de OpenSocial".

Código 18 – Código básico para geração de aplicação em OpenSocial

```
0
```

A API em JavaScript do OpenSocial inclui dois *namespaces*[16]: `opensocial.*` e `gadgets.*`. O primeiro diz respeito a classes e funções relacionadas à configuração do aplicativo. O segundo, possui as classes e métodos mais específicos do aplicativo, que dizem respeito a visualização e mensagens. A API em JavaScript do OpenSocial possui dois "personagens": *OWNER e VIEWER*. *OWNER* refere-se ao dono da aplicação, aquele que a registrou. *VIEWER* refere-se ao usuário que está vendo a aplicação no momento em que ela é executada. No OpenSocial, os aplicativos costumam ser chamados de *Gadgets*. Esse termo refere-se à pequenos *softwares* que podem ser agregados em um ambiente maior. Nos Códigos 19 e 20 há um exemplo de aplicativo OpenSocial que lista os amigos do usuário visualizados da aplicação.

Código 19 – Aplicação em OpenSocial para Listagem de Amigos

```
0
```

Código 20 – Aplicação em OpenSocial para Listagem de Amigos - JavaScripts

```
0
```

O Código 19 é um XML com a configuração do aplicativo. Os JavaScripts foram separados no Código 20 para facilitar a visualização. Neste exemplo, existem algumas tags e atributos não presentes no exemplo do Código 18. A tag `Require` indica ao recipiente quais bibliotecas o aplicativo irá usar, e qual a versão delas. Logo, para criar um aplicativo OpenSocial é necessário informar que este será usado. Os atributos de `ModulePrefs`, `author_mail`, `screenshot` e `thumbnail` contêm o e-mail do autor, uma captura de tela do aplicativo e uma miniatura, respectivamente. Esses atributos são obrigatórios em alguns recipientes. Dentro da tag `Content` há duas tags `div` (titulo e amigos) que serão usadas pelo JavaScript. No JavaScript (Código 20) a requisição é configurada e feita pela função `loadFriends`. Após a conclusão da requisição, a função `onLoadFriends` é chamada. Tal função monta uma lista HTML com os amigos do usuário visualizador e um título com o nome do usuário visualizador. O resultado pode ser visto na Figura 2, para os recipientes Orkut (em cima) e MySpace.

[16]No JavaScript, *namespaces* são pacotes de classes e funções relacionadas.

Figura 2: Visualização de aplicação OpenSocial no Orkut (em cima) e MySpace.

Para facilitar a criação e adaptação de novos recipientes para o OpenSocial, a Apache criou o Shindig[13]. Trata-se de um projeto de código fonte aberto para referência na implementação dos padrões OpenSocial. Ele contem tanto o código fonte para o servidor quanto o do cliente. O Shindig também pode ser usado para testes locais de aplicativos OpenSocial.

3.6 OpenID

O OpenID, em sua versão 1.0, possuía a funcionalidade de um protocolo, para autenticação de URLs, baseado em HTTP. Seu objetivo inicial é centralizar a identificação de usuários para autenticação única em vários web sites. A partir da versão 2.0, o OpenID tornou-se um arcabouço. Com isso, além do protocolo para autenticação, o OpenID passou a integrar um protocolo de transferência de dados e extensões para troca de dados de perfis e mensagens usuário-para-usuário. A Figura 3 mostra um diagrama em pilha do OpenID. O diagrama mostra um arranjo de tecnologias separadas da seguinte maneira:

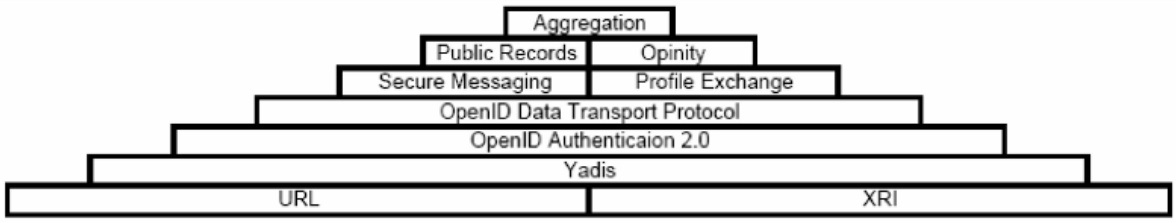

Figura 3: Diagrama em pilha do arcabouço OpenID.

- URLs e XRIs[17] formam a base, representando a identificação dos usuários finais;
- A camada Yadis[18] provê a funcionalidade de uma simples descoberta de serviços, usando o formato XRDS[19] aprovado pela OASIS;
- *OpenID Authentication 2.0* provê uma camada básica para autenticação unificada, da qual serviços de alto nível dependem.

No topo da pilha, outros serviços baseados na identificação, como mensagens seguras ou troca de perfis podem ser colocados em camadas, dependendo das necessidades específicas da implementação [31].

A identificação feita no OpenID é baseada em endereços. Os usuários são identificados por um endereço resolvível (como URLs ou XRIs). Como o OpenID surgiu em uma época em que blogs eram comuns, mesmo que URLs não possuam o foco principal de identificação, os URLs de blogs acabaram sendo usados como identificador de seus autores. Entretanto, o XRI pode ser uma alternativa independente de protocolo, persistente e com privacidade protegida para qualquer identificação [31].

Uma vez que o usuário já possui um endereço digital OpenID, o próximo passo é descobrir quais serviços a identidade do usuário está associada. Para tal é usado o protocolo Yadis. Com ele é possível requisitar um documento XRDS que descreve os recursos usados por um determinado URL ou XRI. O Código 21 mostra um exemplo de documento XRDS. Para identidades ou serviços identificados com XRIs, é necessário registrar o XRI na XDI.org, uma organização sem fins lucrativos e com confiança pública para organização de endereços XDI e XRI.

Código 21 – Exemplo de documento XRDS retornado pelo Yadis.

```
0
```

Na etapa de autenticação, são realizadas uma série de comunicações entre o site em que o usuário está se autenticando, chamado de *Relying Party*, e o provedor do serviço OpenID do

[17] XRI (*eXtensible Resource Identifier*) é um formato de resolução para identificadores abstratos, compatível com URLs. Possui algumas vantagens em relação aos URLs e URIs. Dentre elas destacam-se: o fato de não serem limitados ao protocolo HTTP; distingue acessos a metadados de acessos aos dados do recurso; podem definir um mecanismo de resolução confiável, independente de DNS e podem distinguir entre identificadores persistentes e reatribuídos [32].

[18] Yadis é um protocolo de comunicação para descoberta de serviços. Costuma ser usado para descoberta de serviços para identidades digitais [32].

[19] XRDS (*eXtensible Resource Descriptor Sequence*) é um formato, baseado em XML, para descoberta de metadados acerca de um recurso. Um site com autenticação OpenID, pode resolver a identificação de um usuário, a partir de um documento XRDS, afim de descobrir a localização do provedor do serviço [32].

usuário, chamado *Identity Provider*. As comunicações são realizadas de acordo com o seguinte fluxo [31]:

1. **O usuário** entra com o URL da ***Relying Party (RP)*** para iniciar o processo de autenticação;
2. **RP** busca pelo documento Yadis com o identificador provido pelo **usuário**;
3. **RP** cria uma palavra chave secreta compartilhada com o ***Identity Provider (IdP)*** encontrado por meio da descoberta (passo 2);
4. **RP** redireciona o navegador para o **IdP** encontrado por meio da descoberta (passo 2);
5. **O usuário** entra com seu login e senha no **IdP** e completa o pedido de confiança;
6. **IdP** redireciona o usuário para a **RP** com uma prova criptografada de que o usuário é dono do URL e de qualquer dado de perfil que o usuário escolher liberar;
7. **O Usuário** é redirecionado para a **RP** e agora está autenticado.

O OpenID pode ser totalmente descentralizado. Nenhuma autoridade centralizadora é necessária para aprovar o usuário, o site consumidor (RP) ou o provedor da identidade (IdP). Usando uma funcionalidade chamada *"delegation"*, um usuário pode preservar seu identificador OpenID público, mesmo que ele troque de provedor de identificação (IdP) [31].

Com a introdução do *OpenID Authorization 2.0*, os usuários passaram a ter a opção de usar um endereço digital privado. Este endereço pode identificá-los localmente, em um consumidor (RP) específico, e não no contexto público. O protocolo de transferência de dados, também presente na versão 2.0, provê um método abstrato para intercâmbio de dados entre os provedores de identificação e os consumidores.

4 Considerações Finais

As redes sociais *online* têm crescido e ganhado grande popularidade na Web. Este conceito gerou um novo foco para a Web: a sociabilização a partir de relacionamentos *online*. Além das funcionalidades para busca de amigos e familiares *online*, os sites de redes sociais têm sido usado como um novo meio de comunicação.

Atualmente, grandes redes sociais *online* não podem ser consideradas apenas sites. São plataformas para aplicações sociais. É notável que, mesmo que as aplicações não possuam inteligência, possuem funcionalidades não implementáveis em aplicações normais. Isso se dá por que, em aplicações sociais, os usuários fazem parte da aplicação. De forma que quanto mais usuários, melhor ela é. Um conceito chamado de inteligência coletiva [25].

A partir do estudo de APIs de redes sociais *online*, presente neste trabalho, é possível notar as várias possibilidades para desenvolvimento de aplicações sociais. Tecnologias como OAuth, OpenID e OpenSocial, em conjunto, podem construir aplicações poderosas com interoperabilidade entre redes sociais *online*, segurança na autorização e publicação de dados e comodidade na autenticação de usuários. Aliar isso ao poder da Web Semântica pode ser um estudo para trabalhos futuros.

5 Agradecimento

À Profa. Dra. Deller James Ferreira, pela avaliação do presente texto e pelas sugestões feitas, as quais muito contribuíram para a melhoria do texto original.

Referências

[1] **Introducing JSON**. `http://www.json.org/`, acessado em Março de 2010, 2009.

[2] Bausch, P.; Bumgardner, J.; Fake, C. **Flickr Hacks: Tips & Tools for Sharing Photos Online (Hacks)**. O'Reilly Media, Inc., 2006.

[3] Boyd, D. M.; Ellison, N. B. **Social network sites: Definition, history, and scholarship**. *Journal of Computer-Mediated Communication*, 13(11), 2007.

[4] Cole, C.; Russell, C.; Whyte, J. **Building OpenSocial Apps: A Field Guide to Working with the MySpace Platform**. Addison-Wesley Professional, 2009.

[5] Cole, C.; Russell, C.; Whyte, J. **Building OpenSocial Apps**. Pearson Education, Inc., Boston, MA, US, 2010.

[6] Facebook. **Facebook developers - facebook markup language (fbml)**. `http://developers.facebook.com/docs/reference/fbml/`, acessado em janeiro de 2011, 2011.

[7] Facebook. **Facebook developers - facebook query language (fql)**. `http://developers.facebook.com/docs/reference/fql/`, acessado em janeiro de 2011, 2011.

[8] Facebook. **Facebook developers - permissions**. `http://developers.facebook.com/docs/authentication/permissions`, acessado em janeiro de 2011, 2011.

[9] Fielding, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. PhD thesis, UNIVERSITY OF CALIFORNIA, 2000.

[10] Fitton, L.; Gruen, M.; Poston, L. **Twitter For Dummies**. For Dummies, 2009.

[11] Flickr. **App garden - api documentation**. `http://www.flickr.com/services/api/`, acessado em janeiro de 2011, 2011.

[12] Flickr. **App garden - api documentation - flickr.photos.search**. `http://www.flickr.com/services/api/flickr.photos.search.html`, acessado em janeiro de 2011, 2011.

[13] Foundation, T. A. S. **Apache shindig**. Apache Incubator, 2010.

[14] Fragoso, S. **Wtf a crazy brazilian invasion**. *Fifth International Conference on Cultural Attitudes Towards Technology and Communication*, 1:255–274, 2006.

[15] Gjoka, M.; Kurant, M.; Butts, C. T.; Markopoulou, A. **Walking in facebook: a case study of unbiased sampling of osns**. In: *INFOCOM'10: Proceedings of the 29th conference on Information communications*, p. 2498–2506, Piscataway, NJ, USA, 2010. IEEE Press.

[16] Grewe, L. **OpenSocial Network Programming**. Wrox Press Ltd., Birmingham, UK, UK, 2009.

[17] Hammer-Lahav, E. **The authoritative guide to oauth 1.0**. *Hueniverse*, 2009.

[18] Hammer-Lahav, E. **Introducing oauth 2.0**. *Hueniverse*, 2010.

[19] Hammer-Lahav, E. **The oauth 1.0 protocol**. *Internet Engineering Task Force (IETF)*, 2010.

[20] Hammer-Lahav, E.; Recordon, D.; Hardt, D. **The oauth 2.0 protocol**. *Network Working Group*, 2010.

[21] Henderson, C.; Cope, A. S.; Costello, E.; Mourachov, S.; Butterfield, S. **Flickr authentication api**. *Flickr App Garden*, 2010.

[22] Horwitz, R. **Google launches opensocial to spread social applications across the web**. *Google Press Center*, 2007.

[23] Krishnamurthy, B.; Gill, P.; Arlitt, M. **A few chirps about twitter**. In: *WOSP '08: Proceedings of the first workshop on Online social networks*, p. 19–24, New York, NY, USA, 2008. ACM.

[24] Kwak, H.; Lee, C.; Park, H.; Moon, S. **What is twitter, a social network or a news media?** In: *WWW '10: Proceedings of the 19th international conference on World wide web*, p. 591–600, New York, NY, USA, 2010. ACM.

[25] Lykourentzou, I.; Vergados, D. J.; Loumos, V. **Collective intelligence system engineering**. In: *MEDES '09: Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, p. 134–140, New York, NY, USA, 2009. ACM.

[26] MacKinnon, I.; Warren, R. **Age and geographic inferences of the livejournal social network**. In: *ICML'06: Proceedings of the 2006 conference on Statistical network analysis*, p. 176–178, Berlin, Heidelberg, 2007. Springer-Verlag.

[27] Makice, K. **Twitter API: Up and Running Learn How to Build Applications with the Twitter API**. O'Reilly Media, Inc., 2009.

[28] Negoescu, R.-A.; Adams, B.; Phung, D.; Venkatesh, S.; Gatica-Perez, D. **Flickr hypergroups**. In: *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, p. 813–816, New York, NY, USA, 2009. ACM.

[29] Neville, P. **LinkedIn Top Success Secrets and Best Practices: LinkedIn Experts Share The World's Greatest Tips**. Emereo Pty Ltd, London, UK, UK, 2008.

[30] Reagan, D. **Twitter Application Development For Dummies**. For Dummies, 2010.

[31] Recordon, D.; Reed, D. **Openid 2.0: a platform for user-centric identity management**. In: *DIM '06: Proceedings of the second ACM workshop on Digital identity management*, p. 11–16, New York, NY, USA, 2006. ACM.

[32] Reed, D.; Chasen, L.; Tan, W. **Openid identity discovery with xri and xrds**. In: *IDtrust '08: Proceedings of the 7th symposium on Identity and trust on the Internet*, p. 19–25, New York, NY, USA, 2008. ACM.

[33] Rivlin, G. **Wallflower at the web party**. *The New York Times*, Outubro 2006.

[34] Tyagi, S. **Restful web services**. `http://java.sun.com/developer/technicalArticles/WebServices/restful/`, acessado em maio de 2010, 2006.