David Marquês Francisco

José António Capela Dias

Sistemas Distribuídos (1º Semestre - 2009/2010)

Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia

Universidade de Coimbra

Índice

Introdução

O projecto descrito neste relatório consiste numa aplicação cliente, executada na linha de comandos, que comunica com os servidores do serviço *Twitter.com* através da utilização de *REST* e da *API* pública disponibilizada pelo referido serviço. A esta aplicação foi dada o nome de *MyTwitter*.

Esta provém da evolução de uma série de fases em que foram utilizadas diferentes tecnologias, desde ligações simples através da utilização de *sockets TCP*, passando pelo *RMI* e pelo *Java NIO*. Contrariamente às versões anteriormente desenvolvidas, na nova aplicação não houve desenvolvimento de nenhum servidor, visto que este é oferecido livremente por intermédio de uma *API*. Foram implementadas duas formas de autenticação, *Basic* e *OAuth*, podendo o utilizador optar pela utilização de uma destas. O programa *MyTwitter* foi registado no site *Twitter.com* sob o nome *MyTweet DEI-FCTUC*.

As funcionalidades oferecidas são semelhantes às existentes nas versões anteriores, permitindo ao utilizador interagir com outras pessoas através do envio e recepção de *tweets* e seguir ou ser seguido de forma a manter-se ligado a outros utilizadores do sistema. Actualmente, o *Twitter* é uma das redes sociais mais bem sucedidas da *Web 2.0*.

Detalhes da Aplicação Cliente

A aplicação cliente da nova meta encontra-se estruturada de forma a que haja uma clara divisão entre a interface de interacção com o utilizador, a realização de pedidos ao servidor e o *parsing* das respostas obtidas. Segue-se um diagrama *UML* de pacotes que pretende dar uma visão geral da organização deste *software*.

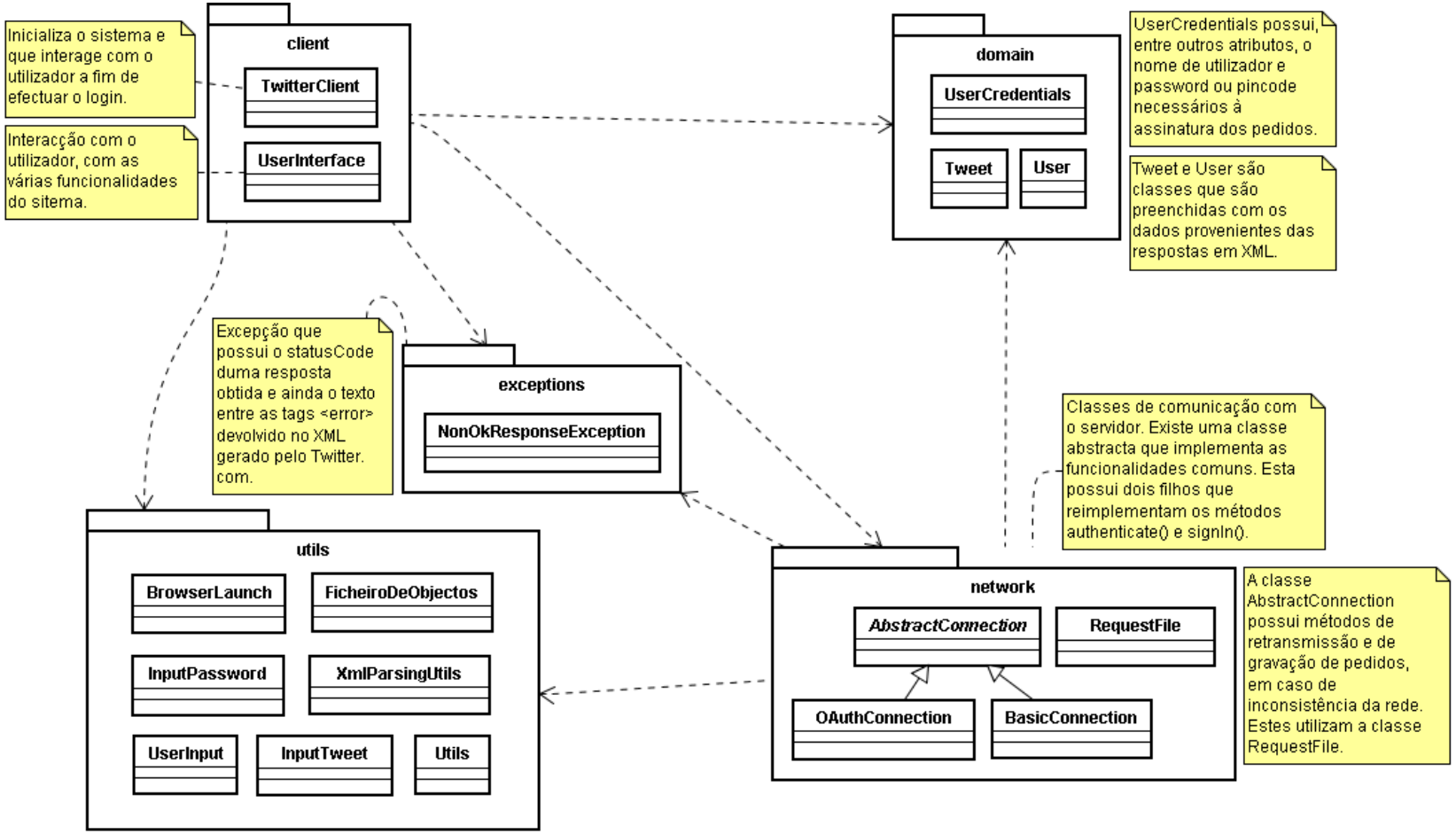


Fig.1 Diagrama *UML* de pacotes do código da aplicação.

Interacção com o utilizador

No que diz respeito à interacção com o utilizador, a nova versão deste projecto mantém-se praticamente inalterada. As diferenças mais notáveis são relativas ao tratamento de excepções pois, contrariamente às fases anteriores, em que era necessário manter *back-compability* com versões que utilizavam tecnologias diferentes, nesta nova versão isso não acontece. Assim sendo, existe uma maior liberdade no lançamento e tratamento destas (queremos com isto dizer que, como na fase anterior, as funções do servidor utilizadas eram as mesmas, quer para *RMI*, quer para *NIO*, não era possível fazer uma partilha limpa de excepções entre as duas aplicações). A aplicação é iniciada pela classe *TwitterClient* que pede ao utilizador o seu *login* e *password*, ou *pincode*, conforme o tipo de autenticação especificado (este processo é explicado mais à frente neste documento). Após realizar o *signin*, o utilizador interage com uma instância da classe *UserInterface*, a qual possui os vários métodos associadas as funcionalidades do sistema.

Segue-se uma parte do *JavaDOC* associado aos métodos da classe *TwitterClient*.

private UserCredentials	**askLogin**() Pede o *username* e *password*, ou *pincode*, ao utilizador.
static void	**main**(String[] args) Recebe da consola o tipo de autenticação que pretende utilizar.
private UserCredentials	**register**() Função não implementada.
private int	**showMenu**() Mostra o menu com as várias opções existentes e retorna a opção escolhida.
private UserCredentials	**signIn**() Chama *askLogin()* e inicia a sessão no servidor de *Twitter*.
void	**start**()
private void	**startTwitter**(UserCredentials user) Permite utilizar as várias funcionalidades do *Twitter*, isto é, executa a classe *UserInterface* (esta não é uma nova *thread*).

De seguida é apresentado o *JavaDOC* associado aos métodos da classe *UserInterface*.

private void	**displayAllTweets**() Mostra uma lista ordenada cronologicamente de todos os *tweets* dos utilizadores que segue, incluindo os *tweets* enviados pelo próprio utilizador.
private void	**displayFollowers**() Mostra os utilizadores que seguem determinado utilizador.
private void	**displayFollowing**() Mostra os utilizadores que determinado utilizador segue.

private void	displayUserProfile() Mostra o *profile* de um utilizador.
private void	displayUserTweets() Mostra os *tweets* de um utilizador.
private void	followUser() Pede o *screen name* do utilizador cujos *tweets* pretendemos seguir.
private void	newTweet() Envia um *tweet* (máximo 140 caracteres; é feita a truncagem) que poderá ser lido pelo próprio e por todos os que seguem este utilizador.
private void	printConnectionProblem() Imprime um aviso de que a ligação está lenta ou o servidor não responde, caso tenha sido notificado pela classe de comunicação com o servidor.
private void	searchUser() Realiza, na lista de todos os utilizadores (existente no servidor), uma pesquisa em busca de uma correspondência total ou parcial do nome introduzido.
private int	showMenu() Mostra o menu com as várias opções existentes e retorna a opção escolhida.
private void	signOut() Faz o *logout* da aplicação.
void	start(UserCredentials user, AbstractConnection conn)
private void	unfollowUser() Pede o *screen name* do utilizador cujos *tweets* pretendemos deixar de seguir.

Comunicação com o servidor

Quanto à comunicação com o servidor, isto é, ao envio de pedidos e recepção de respostas, a aplicação cliente da nova meta possui um objecto da classe *AbstractConnection* que é responsável pela comunicação com a *API* do serviço *twitter.com*. Tal como foi referido, uma vez ligado, o utilizador pode seleccionar opções presentes na *UserInterface*. Esta classe irá pedir ao utilizador os parâmetros de acordo com a opção escolhida, caso existam. De seguida, invoca o método correspondente da classe *AbstractConnection* com os parâmetros inseridos e esta trata de comunicar com a *API* e de receber a resposta ao pedido efectuado. Após ser realizado o *parsing* da resposta, o resultado é devolvido à *UserInterface* que irá mostrar ao utilizador uma mensagem ao utilizador, conforme o resultado recebido.

A classe *AbstractConnection* possui as várias funções necessárias à realização dos pedidos, à sua retransmissão e armazenamento em caso de falha na rede. Possui apenas duas funções que devem ser reimplementadas pelas classes que dela herdam, isto é, *BasicConnection* e *OAuthConnection*, que são a função de assinatura dos pedidos e o método inicial de *signin* do cliente.

Segue-se o *JavaDOC* da classe *AbstractConnection*.

static List<NameValuePair>	**addParameters**(HttpEntityEnclosingRequestBase request, BasicNameValuePair... pairs) Adicionar parâmetros a um pedido do tipo *post*.
***abstract* void**	**authenticate**(HttpUriRequest request,UserCredentials user) Realiza autenticação de tipo *BASIC* ao pedido passado por parâmetro.
HttpResponse	**doPersistentRequest**(HttpUriRequest request, List<NameValuePair> params, UserCredentials author, int nattempt, int waitry) Realiza um pedido (*get* ou *post*) e tenta a sua retransmissão em caso de falha. Se a ligação estiver em baixo, armazena o pedido num ficheiro em disco.
HttpResponse	**doRequest**(HttpUriRequest request) Realiza um pedido (*get* ou *post*) e devolve a resposta obtida
HttpResponse	**doRequest**(HttpUriRequest request,UserCredentials author, boolean pending) Realiza um pedido (*get* ou *post*) e devolve a resposta obtida. Se *pending* estiver a *true*, envia pedidos que estejam pendentes.
HttpResponse	**doRequest**(HttpUriRequest request,UserCredentials author, int nattempt, int waitry, boolean pending) Realiza um pedido (*get* ou *post*) e tenta a sua retransmissão em caso de falha.
void	**followUser**(UserCredentials user, String follow) Segue um utilizador com determinado 'screen name'.
ArrayList<User>	**getFollowers**(UserCredentials user, String username) Mostra os utilizadores que seguem determinado utilizador.
ArrayList<User>	**getFollowing**(UserCredentials user, String username) Mostra os utilizadores que um determinado utilizador segue.
ArrayList<Tweet>	**getTweets**(UserCredentials user) Mostra uma lista ordenada cronologicamente de todos os *tweets* dos utilizadores que segue, incluindo os *tweets* enviados pelo próprio utilizador.
User	**getUserProfile**(UserCredentials user, String username) Mostra o *profile* de um utilizador.
ArrayList<Tweet>	**getUserTweets**(UserCredentials user, String username) Mostra os tweets de um utilizador.
void	**init**()
abstract boolean	**register**(UserCredentials user) Função não implementada.
ArrayList<User>	**searchUser**(UserCredentials user, String query) Realiza, na lista de todos os utilizadores (existente no servidor), uma pesquisa em busca de uma correspondência total ou parcial do nome introduzido.

void	**sendTweet**(UserCredentials user, String tweet) Envia um *tweet* (máximo 140 caracteres; é feita a truncagem) que poderá ser lido pelo próprio e por todos os que seguem este utilizador.
abstract **boolean**	**signIn**(UserCredentials user) Inicia a sessão no servidor de *Twitter*.
void	**signOut**(UserCredentials user) Faz o *logout* da aplicação.
void	**terminate**()
void	**unfollowUser**(UserCredentials user, String unfollow) Pede o *screen name* do utilizador cujos *tweets* pretendemos deixar de seguir.
static void	**useExpectContinue**(HttpUriRequest request) Assegurar a reutilização do *socket*.

Seguem-se alguns métodos da classe *AbstractConnection*, comuns para qualquer tipo de autenticação.

```
/** Assegurar a reutilização do socket */
public void useExpectContinue(HttpRequestBase request)
{
     HttpParams params = new BasicHttpParams();
     HttpProtocolParams.setUseExpectContinue(params, false);
     request.setParams(params);
}
```

Este método é utilizado para assegurar a reutilização do socket, sendo para isso necessário colocar o parâmetro *UseExpectContinue* com o valor *false*.

```
/** Adicionar parâmetros a um pedido do tipo POST */
public static List<NameValuePair> addParameters(HttpEntityEnclosingRequestBase
     request, BasicNameValuePair ... pairs)
{
     List<NameValuePair> formparams = new ArrayList<NameValuePair>();
     for (BasicNameValuePair pair : pairs)
           formparams.add(pair);

     try { request.setEntity(new UrlEncodedFormEntity(formparams, ENCODING)); }
     catch (UnsupportedEncodingException e) { e.printStackTrace(); }
     return formparams;
}
```

Este método permite adicionar um número variável de parâmetros ao pedido *HTTP*.

Segue-se ainda outro exemplo, de um método de assinatura de pedidos da classe *BasicConnection*, filha da classe acima mencionada.

```
/** Realiza autenticação de tipo BASIC ao pedido passado por parâmetro */
public void authenticate(HttpRequestBase request, UserCredentials user)
{
     String credentials = new BASE64Encoder().encode(
           (user.getName() + ":" + user.getPassword()).getBytes());
     request.setHeader("Authorization", "Basic " + credentials);
}
```

De seguida serão explicados os passos associados às funcionalidades da aplicação.

Envio de tweets

Depois de o utilizador inserir o *tweet* desejado, a classe *UserInterface* chama o método correspondente ao envio de *tweets* da classe *AbstractConnection*, ou seja, o método *sendTweet*:

```
/**
 * Envia um tweet (máximo 140 caracteres; é feita a truncagem) que poderá ser
 * lido pelo próprio e por todos os que seguem este utilizador. Cada tweet tem
 * a data em que foi digitado (no lado do cliente).
 */
private void newTweet()
{
	System.out.println("What are you doing?");
	String msg = InputTweet.readTweet();
	if (msg.isEmpty()) {
		System.out.println("Empty tweets are not cool."); return;
	}
	try {
		conn.sendTweet(user, msg);
		// O tweet foi adicionado no servidor
		System.out.println("Tweet sent successfully.");

	} catch (NonOkResponseException e) {
		System.out.println(e.getFriendlyMessage());
	} catch (IOException e) {
		printConnectionProblem();
	}
}
```

O envio de um *tweet* consiste na criação de um objecto *HttpPost* e na adição de alguns parâmetros a esse objecto, nomeadamente os parâmetros de autenticação, *useExpectContinue* e o parâmetro *status* que corresponde à mensagem do *tweet*. Para tal são usadas as funções *authenticate*, *useExpectContinue* e *addParameters* já descritas anteriormente. Segue-se o código responsável por esse processo:

```
/**
 * Envia um tweet (máximo 140 caracteres; é feita a truncagem) que poderá ser
 * lido pelo próprio e por todos os que seguem este utilizador.
 * @see apiwiki.twitter.com/Twitter-REST-API-Method%3A-statuses%C2%A0update
 */
public void sendTweet(UserCredentials user, String tweet) throws IOException
{
	String url = url_statuses + "update.xml";
	HttpPost request = new HttpPost(url);
	useExpectContinue(request);
	List<NameValuePair> p = addParameters(request, new BasicNameValuePair("status",
		tweet));

	authenticate(request, user);
	HttpResponse response = doPersistentRequest(request, p, user, N_ATTEMPT, WAIT_TRY);
	response.getEntity().consumeContent();
}
```

Entrada de um cliente (signIn) com autenticação BASIC

Após a inserção do *username* e da *password*, é feito o pedido de *signIn*, ou seja é chamado o método *signIn* da instância *Connection*:

```
/** Inicia a sessão no servidor de Twitter */
private UserCredentials signIn()
{
     UserCredentials user = askLogin(); // Pede username e password desejados
     if (user == null) return null;

     try {
           if (conn.signIn(user))
                 return user;
     } catch (NonOkResponseException e) {
           System.out.println(e.getFriendlyMessage());
     } catch (IOException e) {
           System.out.println("We're having problems connecting to server...")
              + e.getLocalizedMessage());
     }
     return null;
}
```

O pedido de *signIn* consiste na criação de um pedido *HttpGet* e na adição dos parâmetros de autenticação. Segue-se o código responsável por esse processo:

```
/**
* Inicia a sessão no servidor de Twitter
* @api apiwiki.twitter.com/Twitter-REST-API-Method:-account%C2%A0verify_credentials
*/
public boolean signIn(UserCredentials user) throws IOException
{
     String url = url_account + "verify_credentials.xml";
     HttpGet request = new HttpGet(url);
     authenticate(request, user);

     HttpResponse response = doRequest(request, user, N_ATTEMPT, WAIT_TRY, true);
     String res = XmlParsingUtils.parseUsername(response);
     return res != null;
}
```

Se os dados inseridos estiverem correctos, então será mostrado ao utilizador o menu com as várias opções existentes. Caso contrário o utilizador será notificado de que os dados não estão correctos e voltará ao menu inicial.

Entrada de um cliente (signIn) com autenticação OAuth

Utilizando autenticação *OAuth* é necessário em primeiro lugar obter um *token* de acesso ao *Service Provider*. De seguida, associamos os atributos *accessToken* e *tokenSecret* à instância da classe *UserCredentials*. Posteriormente, é criada uma instância da classe *HttpURLConnection* que através da função *openConnection* irá representar a ligação ao objecto remoto ao qual o *URL* se refere. Depois, o pedido é autenticado através da função *sign*. Por último, é aberta a ligação e enviado o pedido através do método *connect*.

Segue-se o código responsável por esse processo:

```
/**
* Inicia a sessão no servidor de Twitter
* @api apiwiki.twitter.com/Twitter-REST-API-Method:-account%C2%A0verify_credentials
 */
public boolean signIn(UserCredentials user) throws IOException
{
     try {
            provider.retrieveAccessToken(user.getPassword());

            String accessToken = consumer.getToken();
            String tokenSecret = consumer.getTokenSecret();
            user.setAccessToken(accessToken);
            user.setTokenSecret(tokenSecret);
            consumer.setTokenWithSecret(accessToken, tokenSecret);

            URL url = new URL(url_account + "verify_credentials.xml");
            HttpURLConnection request = (HttpURLConnection) url.openConnection();
            consumer.sign(request);

            request.connect();
            if (request.getResponseCode() == HttpStatus.SC_OK)
                  return true;

     } catch (OAuthNotAuthorizedException e) { // Rejeitou autenticação
            return false;
     } catch (OAuthMessageSignerException e) {
            System.out.println(e.getLocalizedMessage());
     } catch (OAuthExpectationFailedException e) {
            System.out.println(e.getLocalizedMessage());
     } catch (OAuthCommunicationException e) {
            System.out.println(e.getLocalizedMessage());
     }
     return false;
}
```

Se os dados inseridos estiverem correctos, então será mostrado ao utilizador o menu com as várias opções existentes. Caso contrário o utilizador será notificado de que os dados não estão correctos e voltará ao menu inicial.

Envio de requests e recepção de replies

Para cada funcionalidade existente na aplicação, o modo de funcionamento é bastante semelhante. A instância *UserInterface* irá receber o pedido do utilizador e transmiti-lo á instância *Connection* que irá criar o pedido *HTTP* adequado à funcionalidade em questão (*HttpPost* ou *HttpGet*) e irá associar-lhe o *URL* correspondente, assim como os parâmetros necessários. O objecto *Connection* (instanciado por *BasicConnection* ou *OAuthConnection*) irá receber a resposta e é feito de seguida o *parsing* dessa resposta. Por último, depois do *parsing*, a resposta será devolvida à *UserInterface* e será apresentada ao utilizador.

Vejamos o exemplo da funcionalidade *Search*:

```
/**
 * Realiza, na lista de todos os utilizadores (existente no servidor), uma
 * pesquisa em busca de uma correspondência total ou parcial do nome introduzido.
 */
private void searchUser()
{
	System.out.print("Who are you looking for? ");
	String searchUser = UserInput.readString();
	if (searchUser.isEmpty()) {
		System.out.println("Invalid query."); return;
	}

	try {
		ArrayList<User> res = conn.searchUser(user, searchUser);
		System.out.println("Name results: ");
		for (User user : res)
			System.out.println("  "+ user);

	} catch (NonOkResponseException e) {
		System.out.println(e.getFriendlyMessage());
	} catch (IOException e) {
		printConnectionProblem();
	}
}
```

Na *UserInterface* é pedido o nome do utilizador a pesquisar e é invocado o método *searchUser* da instância *Connection:*

```
/**
* Realiza, na lista de todos os utilizadores (existente no servidor), uma
* pesquisa em busca de uma correspondência total ou parcial do nome introduzido.
* @api apiwiki.twitter.com/Twitter-REST-API-Method%3A-users-search
*/
public ArrayList<User> searchUser(UserCredentials user, String query)
	throws IOException
{
	String url = url_users + "search.xml?q=" + query.trim();
	HttpGet request = new HttpGet(url);
	authenticate(request, user);

	HttpResponse response = doRequest(request, user, N_ATTEMPT, WAIT_TRY, true);
	return XmlParsingUtils.parseUsers(response);
}
```

A instância *Connection* irá juntar o nome inserido pelo utilizador ao *URL* correspondente à funcionalidade *search* e irá criar um pedido *HttpGet* com esse *URL*. De seguida irá tratar da autenticação desse pedido e do seu envio e recepção. Posteriormente, executa o *parsing* da resposta obtida e devolve a resposta à *UserInterface* que a irá apresentar ao utilizador.

Saída de um cliente (signOut)

Como não foi estabelecida uma ligação ao servidor, a saída de um cliente consiste apenas numa função vazia que levará o utilizador ao menu inicial da aplicação.

```
/** Termina a sessão do utilizador */
public Reply signOut(Request request)
{
}
```

Persistência de dados

Em caso de falha na rede, são realizadas várias tentativas de reenvio.

```
protected static final int N_ATTEMPT = 5; // Número de retransmissões
protected static final int WAIT_TRY = 500; // Intervalo de tempo entre elas
```

Se após essas várias tentativas a rede continuar em baixo, os pedidos são armazenados num ficheiro binário em disco. Para esse efeito, é utilizada a classe *RequestFile*. No entanto, coloca-se um problema, que é o facto de os pedidos do tipo *HttpPost* e *HttpGet* não serem serializáveis. Foi por isso criada uma classe auxiliar, igualmente presente no ficheiro *RequestFile.java* que permite construir uma representação serializável do objecto. Segue-se o código associado à referida classe.

```
class Request implements Serializable
{
	private static final long serialVersionUID = 1L;

	private String url;
	private String type;
	private List<NameValuePair> params;

	Request(HttpUriRequest request, List<NameValuePair> params)
	{
		url = "http://" + AbstractConnection.HOSTNAME + request.getURI().getPath();
		this.type = request.getMethod();
		this.params = params;
	}
```

```
	public HttpUriRequest convert2HttpRequest(UserCredentials author)
	{
		HttpUriRequest request;
		if (type.equals(HttpPost.METHOD_NAME))
		{
			request = new HttpPost(url);
			AbstractConnection.useExpectContinue(request);

			try {
				HttpEntity entity = new UrlEncodedFormEntity(params,
					AbstractConnection.ENCODING);
				((HttpPost) request).setEntity(entity);
			} catch (UnsupportedEncodingException e) { }
		}
		else // if (r.type.equals(HttpGet.METHOD_NAME))
			request = new HttpGet(url);

		return request;
	}
}
```

Como se pode observar no código, através do construtor da classe podemos passar um pedido *http* para o formato serializável *Request* e, através do método *convert2HttpRequest*, realizar o processo inverso, isto é, criar um novo pedido *http* a partir dum objecto desta classe.

Os pedidos pendentes de um determinado utilizador são armazenados num ficheiro e posteriormente recarregados e assinados, quando a ligação estiver novamente activa. A gestão dos ficheiros por cada utilizador é feita pela classe *RequestFile*, que possui uma interface bastante simples:

```
	static boolean	append(java.lang.String username, Request request)

	static boolean	delete(java.lang.String username)

	static boolean	exists(java.lang.String username)

	static ArrayList<Request>	read(java.lang.String username)

	static boolean	write(java.lang.String username, ArrayList<Request> requests)
```

Utilização de Apache HTTPClient

Nesta aplicação foi utilizada a biblioteca *Apache HttpClient 4* para realizar os pedidos *http* e a biblioteca *JDOM* como *parser XML* para realizar o *parsing* das respostas como será mostrado no capítulo seguinte.

A realização de pedidos é feita através dos seguintes métodos:

HttpResponse	**doRequest(HttpUriRequest request)** Realiza um pedido (*get* ou *post*) e devolve a resposta obtida.
HttpResponse	doRequest(HttpUriRequest request,UserCredentials author, boolean pending) Realiza um pedido (*get* ou *post*) e devolve a resposta obtida. Se *pending* estiver a *true*, envia ainda pedidos que estejam pendentes.
HttpResponse	doRequest(HttpUriRequest request,UserCredentials author, int nattempt, int waitry, boolean pending) Realiza um pedido (*get* ou *post*) e tenta a sua retransmissão em caso de falha. Se *pending* estiver a *true*, envia ainda pedidos que estejam pendentes.
HttpResponse	doPersistentRequest(HttpUriRequest request, List<NameValuePair> params, UserCredentials author, int nattempt, int waitry) Realiza um pedido (*get* ou *post*) e tenta a sua retransmissão em caso de falha. Envia ainda pedidos que estejam pendentes. Se a ligação estiver em baixo, armazena o pedido num ficheiro em disco.

Segue-se a implementação de cada um dos referidos procedimentos. Como se poderá observar, os métodos chamam-se entre si (de baixo para cima), sendo que cada um acrescenta uma funcionalidade ao método. Apesar de se ter construído esta classe de forma a que a sua utilização pudesse servir para várias problemáticas, no caso em questão, todos os pedidos são enviados através do terceiro método indicado, exceptuando-se o envio de *tweets*, o qual é feito através do quarto método.

```
/** Realiza um pedido (GET ou POST) e devolve a resposta obtida */
public HttpResponse doRequest(HttpUriRequest request) throws IOException
{
     HttpResponse response = httpClient.execute(host, request);
     StatusLine statusLine = response.getStatusLine();

     if (statusLine.getStatusCode() != HttpStatus.SC_OK)
     {
           String twitterErrorMsg = "";
           try { // O twitter possui mensagens de erro específicas que vêm no XML
                 Document d = new
                 SAXBuilder().build(response.getEntity().getContent());
                 twitterErrorMsg = d.getRootElement().getChildText("error");
           }
           catch (IllegalStateException e) { }
           catch (JDOMException e) { }
```

```
            throw new NonOkResponseException(statusLine, twitterErrorMsg);
      }
      return response;
}
```

```
/**
 * Realiza um pedido (GET ou POST) e devolve a resposta obtida. Se 'pending'
 * for 'true', envia também os pedidos que estejam num ficheiro.
 */
public HttpResponse doRequest(HttpUriRequest request, UserCredentials author,
      boolean pending) throws IOException
{
      /* De notar que este método deve ser utilizado somente para enviar pedidos
       * armazenados cuja resposta não é importante (como por exemplo, enviar um
       * tweet) pois a única resposta devolvida é a do pedido recebido por
       * parâmetro */

      // Se se pretende enviar também pedidos que estejam armazenados
      if (pending && RequestFile.exists(author.getName()))
      {
            ArrayList<Request> requests = RequestFile.read(author.getName());

            for (Request r : requests) {
                  HttpUriRequest oldRequest = r.convert2HttpRequest(author);
                  authenticate(oldRequest, author);
                  HttpResponse response = doRequest(oldRequest);
                  response.getEntity().consumeContent();
            }
            RequestFile.delete(author.getName());

            System.out.println("Your pending tweets have been sent.");
      }
      return doRequest(request);
}
```

```
/**
 * Realiza um pedido (GET ou POST) e tenta a sua retransmissão em caso de falha.
 * Se 'pending' for 'true', envia também os pedidos que estejam num ficheiro.
 */
public HttpResponse doRequest(HttpUriRequest request, UserCredentials author,
      int nattempt, int waitry, boolean pending) throws IOException
{
      int i = 1;
      while (true) // Tenta reenviar o pedido 'nattempt' vezes
      {
            try { return doRequest(request, author, pending); } // Tenta enviar
            // Ocorreu erro, mas não de ligação
            catch (NonOkResponseException e) { throw e; }
            catch (IOException e) { if (i == nattempt) throw e; } // Erro de ligação
            i++;
            try { Thread.sleep(waitry); }
            catch (InterruptedException e) { }
      }
}
```

```java
/**
 * Realiza um pedido (GET ou POST) e tenta a sua retransmissão em caso de falha.
 * Se não conseguir enviar o pedido, este é armazenado num ficheiro.
 * @throws IOException, NonOkResponseException
 */
public HttpResponse doPersistentRequest(HttpUriRequest request,
      List<NameValuePair> params, UserCredentials author, int nattempt, int waitry)
      throws IOException
{
      try {
            return doRequest(request, author, nattempt, waitry, true);
      }
      catch (IOException e) {
            RequestFile.append(author.getName(), new Request(request, params));
            throw e;
      }
}
```

No próximo capítulo será feita uma rápida descrição da utilização do *parser* JDOM, de forma a realizar a conversão do código *XML* para as classes *User* e *Tweet* em *Java*.

Utilização de XML Parser

Foi utilizado o *parser JDOM* que possibilita uma representação de fácil e eficiente leitura de documentos *XML* ao possuir uma *API* que é simples e intuitiva. Quando comparada com a biblioteca fornecida no *tutorial* disponibilizado no *WoC*, esta é de mais fácil utilização e é mais elegante; no entanto, como será visto, não é feita uma leitura sequencial e única do ficheiro *XML*, mas sim várias leituras com base na *query* fornecida, o que diminui a sua performance. Veja-se o exemplo do *parsing* de um conjunto de *tweets*:

```
Document d = new SAXBuilder().build(response.getEntity().getContent());

List<Element> statuses = d.getRootElement().getChildren("status");
for (Element status : statuses) {
     Tweet t = parseTweet(status);
     t.setAuthor(parseUser(status.getChild("user")));
     tweets.add(t);
}
```

Para cada *tag status* do *XML* recebido é feito o *parsing* da informação relativa ao *tweet* e é adicionado ao campo *author* do *tweet* o *user* que o escreveu. Ao percorrer a lista *statuses* temos acesso a todos os *tweets* presentes no *XML* e para cada um desses *tweets* são chamadas as seguintes funções:

```
public static Tweet parseTweet(Element e)
{
     Tweet tweet = new Tweet();

     tweet.setId(e.getChildText("id"));
     tweet.setCreatedAt(e.getChildText("created_at"));
     tweet.setText(e.getChildText("text"));
     tweet.setSource(e.getChildText("source"));
     tweet.setTruncated(Boolean.parseBoolean(e.getChildText("truncated")));
     tweet.setReplyToTweetId(e.getChildText("in_reply_to_status_id"));
     tweet.setReplyToUserId(e.getChildText("in_reply_to_user_id"));

     return tweet;
}
```

```
public static User parseUser(Element e)
{
     User user = new User();

     user.setId(e.getChildText("id"));
     user.setName(e.getChildText("name"));
     user.setScreenName(e.getChildText("screen_name"));
     user.setDescription(e.getChildText("description"));
     user.setUrl(e.getChildText("url"));
     user.setProtectedProfile(Boolean.parseBoolean(e.getChildText("protected")));
     user.setFollowers(Integer.parseInt(e.getChildText("followers_count")));
     user.setFollowing(Integer.parseInt(e.getChildText("friends_count")));
     user.setCreatedAt(e.getChildText("created_at"));
     user.setTweets(Integer.parseInt(e.getChildText("statuses_count")));

     return user;
}
```

Estes dois métodos irão passar os conteúdos presentes no ficheiro *XML* para instâncias das classes *Tweet* e *User*, ficando assim essas informações acessíveis de um modo mais prático do ponto de vista do *Java*. Para fazer o *parsing* de um conjunto de utilizadores o mecanismo é semelhante:

```
ArrayList<User> results = new ArrayList<User>();
Document d = new SAXBuilder().build(response.getEntity().getContent());
List<Element> users = d.getRootElement().getChildren("user");
     for (Element user : users)
           results.add(parseUser(user));

return results;
```

Ao percorrer a lista *users*, para cada um dos utilizadores presentes na lista irá ser chamada a função *parseUser* cujo código foi apresentado no exemplo anterior.

Manual do utilizador

Ao entrar na aplicação cliente surge o seguinte menu:

```
 Welcome to MyTwitter!
 Share and discover what’s happening right now, anywhere in the world.
--------------------------------------------------------------------------------
You are connected!
--------------------------------------------------------------------------------
  1. Sign-in
  2. Exit
  C>
```

De seguida serão explicadas cada uma das funcionalidades suportadas pelo programa.

Sign-in / Sign-out

Ao escolher a opção *Sign-in*, caso esteja a utilizar o tipo de autenticação *OAuth*, surgirá o seguinte menu:

```
  1. Sign-in
  2. Exit
  C> 1
Username: UserX
Pin code: 4206066
```

Depois de inserir o nome do utilizador será aberta uma página *web* que lhe pedirá os dados relativos a esse utilizador. Se os dados introduzidos forem válidos, ser-lhe-á atribuído o *Pin Code* a ser inserido para completar o *login*.

Caso esteja a utilizar o tipo de autenticação *Basic*, surgirá o seguinte menu:

```
  1. Sign-in
  2. Exit
  C> 1
Username: UserX
Password: teste
```

Deve inserir o nome do utilizador e a *password* correspondente.

Depois de estar ligado, ser-lhe-ão mostradas várias opções:

```
 Welcome <username>!
--------------------------------------------------------------------------------
  0. Display list of all tweets I am following
  1. Send a Tweet
  2. Search users
  3. Follow a user
  4. Unfollow a user
  5. Display list of followers
  6. Display who am I following
  7. Display user profile
  8. Display tweets from a user
  9. Sign out
  C>
```

Display list of all tweets I am following

Esta opção mostra uma lista ordenada cronologicamente dos vinte *tweets* mais recentes dos utilizadores que segue, incluindo os *tweets* enviados pelo próprio utilizador. *Tweets* não são ordenados pelo remetente, mas sim pela data de envio.

```
"Hello World”
  by User X at Fri Dec 10 17:01:02 +0000 2009.
```

Caso não exista nenhum *tweet* para ser mostrado surgirá a mensagem:

```
There are no tweets to show.
```

Caso ocorra um problema com a sua ligação aparecerá impressa a seguinte informação:

```
Your Internet connection is slow or unavailable.
```

Send a Tweet

Um *tweet* tem no máximo 140 caracteres. Se a sua mensagem tiver um comprimento superior, é feita a sua truncagem.

```
What are you doing (140)?
```

Escolhendo esta opção poderá enviar um *tweet* e receberá a seguinte mensagem:

```
Tweet sent successfully.
```

Caso ocorra algum problema com a sua ligação, será informado desse facto.

```
Your Internet connection is slow or unavailable.
Your tweet has been saved for sending.
```

Search users

```
Username:
```

Ao seleccionar esta opção será realizada, na lista de todos os utilizadores, uma pesquisa de forma a encontrar uma correspondência total ou parcial do nome introduzido. Por exemplo, procurando *Filipe*, são possíveis resultados *Filipe*, *luís_filipe* e *JoãoFilipe*.

Follow a user

```
Username:
```

Escolhendo esta opção poderá seguir um utilizador com um determinado *username*.

```
You are following <username>!
```

Caso esse utilizador não esteja registado no sistema, será avisado desse facto.

```
The specified user does not exist.
```

Caso já esteja a seguir esse utilizador, surgir-lhe-á a seguinte mensagem de aviso:

```
Username: userY
Could not follow user: userY is already on your list.
```

Ao seguir um utilizador, poderá ler os *tweets* por ele publicados. O nome deve ser exactamente igual, mas a comparação é *case-insensitive*.

Unfollow a user

```
Username:
```

Escolhendo esta opção poderá deixar de seguir um utilizador com determinado *username*.

```
You have unfollowed <username>!
```

Caso o utilizador especificado não esteja registado no sistema, ser-lhe-á apresentada a seguinte mensagem de erro:

```
The specified user does not exist.
```

Caso não esteja a seguir esse utilizador, será igualmente avisado desse facto.

```
You are not friends with the specified user.
```

O *username* deve ser igual ao registado no sistema, mas a comparação é *case-insensitive*.

Display list of followers

```
Whose followers list do you want to check ['Enter' if it's yours]:
```

Escolhendo esta opção poderá visualizar os utilizadores que estão a seguir um determinado utilizador. Para visualizar os seus próprios *followers* deve pressionar apenas a tecla *Enter*. Caso pretenda ver a lista de *followers* doutro utilizador deve inserir o *username* desse utilizador. De seguida receberá a mensagem:

```
User userX is being followed by:
user1 (User1)
user2 (User2)
…
```

Caso ninguém esteja a seguir o utilizador especificado, receberá a mensagem:

```
No one is following this user.
```

Display list of friends

```
Whose friends list do you want to check ['Enter' if it's yours]:
```

Escolhendo esta opção poderá visualizar os utilizadores que um utilizador está a seguir. Para visualizar os seus "*friends"* deve pressionar apenas a tecla *Enter*. Caso pretenda ver a lista de *followers* doutro utilizador deve inserir o nome desse utilizador. De seguida receberá a mensagem:

```
User userX is following:
user1 (User1)
user2 (User2)
…
```

Caso ninguém esteja a seguir o utilizador especificado, receberá a mensagem:

```
User userX is following: no one.
```

Display user profile

Esta opção permite-lhe visualizar os dados de um utilizador com um determinado *username*.

```
Who do you want to check? userY
  Full Name       user Y
  Screen Name     userY
  Description
  URL             http://innovation.dei.uc.pt
  Protected       false
  Nº Followers    12
  Nº Following    25
  Created at      Wed Oct 14 18:02:50 +0000 2009
  Tweets    10
```

Display tweets from a user

```
Whose tweets do you want to check ['Enter' if it's yours]:
```

Esta opção permite-lhe visualizar os vinte *tweets* mais recentes de um utilizador com um determinado *username*. Para visualizar os seus próprios *tweets* deve pressionar apenas a tecla *Enter*. Caso pretenda ver a lista de *tweets* de outro utilizador, deve inserir o *username* desse utilizador. Ser-lhe-á posteriormente apresentada a respectiva lista de *tweets*:

```
"Hello World”
  by User Y at Fri Dec 11 14:41:02 +0000 2009.
```

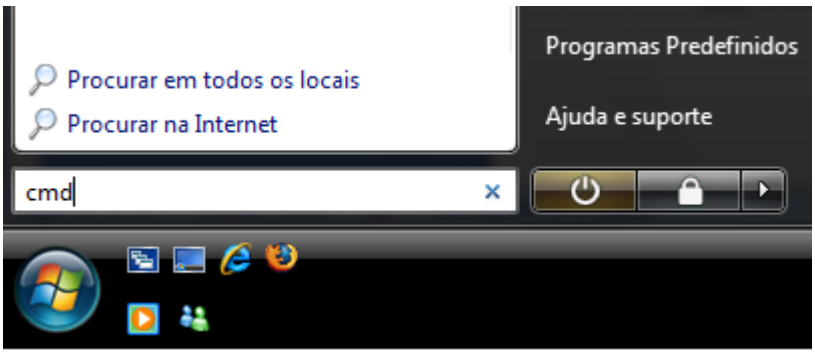
Caso não exista nenhum *tweet* para ser mostrado surgirá a mensagem:

```
There are no tweets to show.
```

Manual de instalação e configuração

Para executar o programa *MyTwitter*, deverá seguir os seguintes passos:

- No menu iniciar, caso esteja a utilizar o *Windows XP*, deve escolher a opção “executar” e digitar o comando ***cmd****;*
- Caso esteja a utilizar o *Windows Vista* ou *Windows 7*, no menu “iniciar procura” deve digitar ***cmd*** e carregar na tecla *Enter* para abrir a linha de comandos;



- Dentro da linha de comandos, deverá navegar até à directoria onde se encontra o ficheiro ***MyTwitterClient.jar***;
- Para executar o programa do cliente deverá digitar o seguinte comando:

  java -jar MyTwitterClient.jar

- Caso pretenda escolher um tipo de autenticação diferente do definido por defeito deve utilizar um dos seguintes comandos:

  java -jar MyTwitterClient.jar OAUTH

  java -jar MyTwitterClient.jar BASIC

Descrição dos testes realizados à aplicação

Display list of all tweets I am following

São mostrados com sucesso os vinte *tweets* mais recentes do utilizador em questão e das pessoas que ele segue.

Send a Tweet

Os *tweets* são enviados ao servidor com sucesso. Os *tweets* são enviados ao servidor quando o cliente realiza o *login*, caso tenha anteriormente ocorrido uma falha.

If the client crashes/restart, the buffered tweets (stored in disk) are dispatched to the server

Se o cliente enviar *tweets* enquanto o servidor está em baixo e fizer *logout*, os seus *tweets* são guardados num ficheiro, e da próxima vez que esse cliente se ligar são enviados para o servidor.

Search users

Foram efectuadas várias pesquisas com vários *username* diferentes e o resultado foi o esperado, ou seja, todos os seus utilizadores cujo *username* continha o nome inserido pelo utilizador apareceram como resultado da pesquisa.

Follow a user/ Unfollow a user

Foram efectuadas várias operações de *follow* e *unfollow* sempre com sucesso.

Display list of followers

Foram efectuados vários pedidos de listas de *followers* e o resultado foi o esperado, ou seja, todos os seguidores apareceram na lista.

Display list of friends

Foram efectuados vários pedidos de listas de *following* e o resultado foi o esperado, ou seja, todas as pessoas que o utilizador seguia surgiram na lista.

Display user profile

Foram efectuados vários pedidos de visualização do perfil de utilizadores e a resposta foi correcta para todos os pedidos.

Display tweets from a user

Foram efectuados vários pedidos de visualização dos *tweets* de um determinado utilizador e a resposta foi correcta para todos os pedidos.

Conclusão

O trabalho prático desenvolvido, não só no que consta desta meta final, mas também às que lhe antecederam, permitiu que consolidássemos conhecimentos basilares relativos à temática dos Sistemas Distribuídos, não só através da implementação de uma arquitectura cliente-servidor, como também na utilização de tecnologias como *Java NIO*, *RMI*, passando por *HTML/JSP* e terminando com o *REST*. O desenvolvimento deste projecto forneceu-nos desta forma novas competências relevantes para a nossa formação e, por isso, consideramos que o trabalho foi simultaneamente interessante e importante.

Autores

David Marquês Francisco nº 2007183509

José António Capela Dias nº 2007183794

Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia
Universidade de Coimbra, 2009/2010