Mobile Ajax for Java ME Technology

Akhil Arora, Senior Staff Engineer, Sun Microsystems Inc. akhil@sun.com

Vincent Hardy, Senior Staff Engineer, Sun Microsystems Inc. vincent.hardy@sun.com

Introduction

There are many Sun Microsystems technologies that use Ajax [Ajax], and more than one way to use Ajax on mobile platforms. For example, applications written using the Java Platform, Enterprise Edition (Java EE, formerly known as J2EE) may generate XML, JSON [JSON], XHTML and/or ECMAScript destined for mobile browsers.

One of the recent advances on the Java Platform, Mobile Edition (Java ME, formerly known as J2ME) is the Mobile Service Architecture [MSA]. MSA is a Java Specification Request (JSR-248) which defines a set of APIs for Java ME which include a wide variety of features, from Bluetooth to payment, multimedia APIs and support for rich, animated graphics.

This paper discusses an effort to provide Java ME developers with tools to create Mobile Ajax applications, combining the simplicity and familiarity of the Ajax programming model with the richness and secure environment of the MSA APIs. This effort takes the form of an open-source library that can be added to any Java ME application. The paper briefly describes this library along with some sample use cases.

Mobile Ajax For The Java ME Platform

Ajax is typically used in the context of Web applications running in a browser and using `XmlHttpRequest` from ECMAScript to retrieve XML or JSON data from RESTful Web Services. The results are applied as updates to the current browser's page DOM (Document Object Model [DOM]).

In the scope of this paper, Mobile Ajax on the Java ME platform is used to mean the following:

- Asynchronous call to the network (using the Mobile Information Device Profile's [MIDP] Generic Connection Framework [GCF]).
- Use of a data serialization format (such as XML or JSON).
- A presentation layer using a DOM-based User Interface (such as XHTML or SVG).

Figure 1 illustrates a typical Mobile Ajax interaction on the Java ME platform.

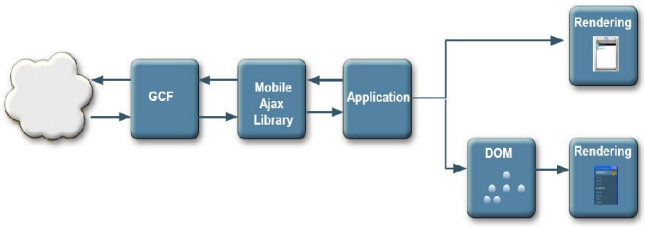

Figure 1: A Java ME Mobile Ajax Interaction

Why Ajax For Java ME Applications?

There are multiple reasons why using the Ajax model is useful in Java ME programs.

Using an Ajax library, the application is simpler and needs only implement synchronous calls or handle callbacks instead of having to deal with the complexities of multi-threaded programming. The library also abstracts low-level data format parsers, which helps reduce application complexity, maintenance and debugging costs.

Web developers can apply their knowledge of the Ajax paradigm to the Java ME platform, reducing the learning curve.

The Java ME platform lets developers leverage the full capabilities of the platform, such as the phone camera, location information, the address book or persistent storage, all of which are not available from a browser environment.

The library's small footprint along with Java ME's strong security architecture and wide deployment make it a robust environment for developing applications using the Mobile Ajax programming model.

Asynchronous Requests

In order to help Java ME developers build Web 2.0 applications more easily, there was a need for a library to easily deal with the following:

- Asynchronous handling of HTTP Get and Post
- Progress callbacks.
- HTTP Basic/Digest Authentication
- URL-encoding
- Multi-part MIME encoding

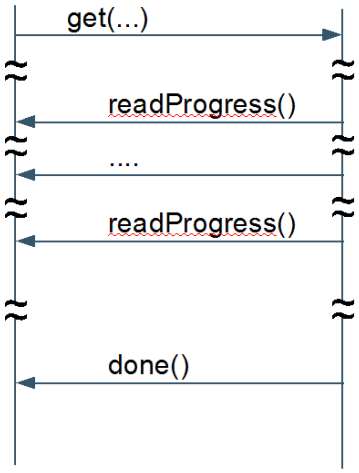

Figure 2: Asynchronous Get Request

An asynchronous HTTP Get call sequence is shown in Figure 2.

The `Request` abstraction provided by the library has the following interface:

```
// synchronous methods

static Response get(String url, Arg[]
inputArgs, Arg[] httpArgs,
ProgressListener listener)

static Response post(String url, Arg[]
inputArgs, Arg[] httpArgs,
ProgressListener listener, PostData
data)

// asynchronous methods

static void get(String url, Arg[]
inputArgs, Arg[] httpArgs,
RequestListener listener, Object
context)

static void post(String url, Arg[]
inputArgs, Arg[] httpArgs,
RequestListener listener, PostData data,
Object context)
```

The `inputArgs` parameter above is used for specifying args that will get URL-encoded, for example:

```
String url = "http://host.com/webapi";
Arg[] args = {
  new Arg("arg1", "val1"),
  new Arg("arg2", "val2")
};
```

The encoded URL becomes "http://host.com/webapi?arg1=val1&arg2=val2".

If the application needs to listen to the progress of a query, for a synchronous or an asynchronous call, it needs to provide an implementation of the `ProgressListener` interface described below:

```
interface ProgressListener {

    void readProgress(Object context,
            int bytes, int total);

    void writeProgress(Object context,
            int bytes, int total);

}
```

Finally, the application gets notified of an asynchronous request result through a callback to

its `RequestListener` implementation's `done()` method:

```
interface RequestListener extends
ProgressListener {

    void done(Object context, Response
result);

}
```

Parsing A Response

In addition to making it easy to generate synchronous and asynchronous requests, the library helps in extracting information from results returned from RESTful Web Services. The results can be in XML or JSON formats. The following paragraphs detail how the Response abstraction is used.

The `Response` class has a few simple methods:

```
class Response {

    // Result contains the parsed
    // returned data
    Result getResult();

    // HTTP response code
    int getCode();

    // HTTP response headers
    Arg[] getHeaders();

    // Exception, if any
    Exception getException();

}
```

The library described in this paper uses a declarative approach similar to XPath to extract information from structured data. XPath was not used because of its large footprint and because it was questionable it would be appropriate to apply XPath expressions to JSON data. The expression language used is very small and simple - it uses ".", "[" and "]" to select specific elements from results. Some examples of paths are:

```
statuses.status[1].text
statuses.status[2].user.screen_name
users.user[3].id
```

The `Result` class provides methods which understand this syntax and return the requested data elements:

```
// accessors for primitive types
boolean getAsBooleanString path);
int getAsInteger(String path);
long getAsLong(String path);
double getAsDouble(String path);
String getAsString(String path);

// accessors for arrays
int getSizeOfArray(String path);
String[] getAsStringArray String path);
int[] getAsIntegerArray(String path);
...
```

The following illustrates how an application may use this API.

```
Result result = response.getResult();
int size =
    result.getSizeOfArray("users.user");
for (int i=0; i < size; i++) {
    String base
      = "users.user[" + i + "].";
    userName =
      result.getAsString(base + "name");
    userId =
      result.getAsInteger(base + "id");
 ...
```

One of the advantages of this approach is that it abstracts the infoset encoding away from the application. The API allows application code to remain unchanged as the underlying encoding is switched between XML or JSON, as shown below:

XML

```
<users>
   <user>
      <name>User 1</name>
   </user>
   <user>
       <name>User 2</name>
   <user>
</users>
```

JSON

```
{ "users":
  {  "user": [
     {    "name": "User 1"  },
     {    "name": "User 2"  }
    ]
  }
}
```

Whichever form of encoding was used, the application can retrieve the data as follows:

```
String name =
result.getAsString("users.user[1].name");
assert "User 2".equals(name);
```

User Interface

On the Java ME platform, there are several APIs which use user interface markup for presentation. The JSR 226 « Scalable 2D Vector Graphics » API, supports rendering, manipulating, interacting with, rendering and playing images in the SVG Tiny 1.1 format. Its follow-on JSR 287 will bring support for SVG Tiny 1.2 and an improved API feature set. Finally, JSR 290 brings support for WICD Mobile Profile 1.0, i.e., XHTML, CSS, SVG and ECMAScript combined.

All these APIs let applications follow the traditional Ajax model and apply the results of synchronous or asynchronous queries to the DOM tree.

Playing a DOM User Interface in a Java ME Application.

The following code snippet illustrates how to load and play an SVG image, using the JSR 226 API.

```
import javax.microedition.m2g.SVGImage;
SVGImage image =
   SVGImage.createImage(url, null);

// Play the image
SVGAnimator animator
= SVGAnimator.createAnimator(image);


Canvas canvas =
(Canvas)animator.getTargetComponent();

getDisplay().setCurrent(canvas);
animator.play();
```

In this model, the Java ME application plays the role of a user agent.

Updating the DOM tree in a Java ME Application,

Let's look at a simple example where the application needs to display the progress of an on-going Ajax query and display an animation when the query is completed.

On the user interface side, this might be done with an SVG images such as the following (this example is simplistic, for the purpose of the explanation, a real-life SVG would have more visual appeal).

```
<svg ...>

<rect id="progress"
 x="20" y="200" width="1"
 height="30" fill="blue"/>

<animateTransform id="doneAnimation"
  attributeName="transform"
  type="translate"
  values="0,0;400,0"
  begin="indefinite"
  dur="0.5s" />

</rect>

<text id="progressText" x="120"
  height="240">0%</text>
</svg>
```

In response to a query, the application could manipulate the DOM tree, for example to scale a rectangle to show the query progress and to start an animation (the "`doneAnimation`") when the request is complete. This is shown below.

```
class ProgressBar implements
                     ProgressListener {
SVGAnimationElement doneAnimation;
SVGLocatableElement progress;
SVGElement progressText;
public ProgressBar(Document doc) {
doneAnimation = (SVGAnimationElement)
doc.getElementById("doneAnimation");
progress = (SVGLocatableElement)
doc.getElementById("progress");
                progressText =
(SVGElement)
doc.getElementById("progressText");
}
```

```
public void readProgress
  (int bytes, int total) {
float pos = (bytes / (float) total);
// Scale the progress bar graphic
SVGMatrix scale =
computeScaleMatrix(pos);
progress.setMatrixTrait("transform",
scale);
progressText.setTrait("#text",
(int) Math.ceil(pos * 100) + "%");
}
void done
  (Object context, Response result) {
doneAnimation.beginElementAt(0);
}
```

Conclusion

For applications that need to use a mobile device's capabilities that are not available in a mobile browser (such as camera, location, bluetooth, etc), a small library that offers the familiar Ajax programming model can ease a Web developer's task.

References

[Ajax] Asynchronous JavaScript And XML, http://en.wikipedia.org/wiki/Ajax_(programming)

[JSON] JavaScript Object Notation, http://en.wikipedia.org/wiki/JavaScript_Object_Notation

[MIDP] Mobile Information Device Profile, http://jcp.org/en/jsr/detail?id=118

[MSA] Mobile Service Architecture, http://jcp.org/en/jsr/detail?id=248

[DOM] Document Object Model, http://www.w3.org/TR/DOM-Level-3-Core/

Resources

Open Source Library for Ajax on Java ME, https://meapplicationdevelopers.dev.java.net/mobileajax.html

Examples of Mobile Ajax Java ME Application, https://meapplicationdevelopers.dev.java.net/phoneme_ui_labs.html