Midlets

Results from a Cocoon pipeline (out of an XML serializer) can also be used to answer a Midlet at a mobile phone (using MIDP/CLDC with the Java 2 Micro Edition or J2ME platform). To do this. the Midlet has to be able to parse the incoming XML, received as a byte array. This message explains the principle to do this.

First problem: How to parse XML in J2ME? There is a solution:
To parse XML in J2ME, use kXML. It works really well! (see: http://kxml.objectweb.org/software/downloads/ to obtain the package and the docs)

How it works

On the server side (Cocoon 2/ Tomcat 4):

Create a pipeline:

1. Action to lookup the data (class, implementing the Action-interface and instatiating an EJB - J2EE to JBoss) and pass the object of interest (delivered by the EJB) to the request with request.setAttribute(<name>, <object>) 2. Generate xsp, using a logicsheet which creates XML from the object 3. Serialize (so skip Transform!) to XML.

On the mobile phone in the Midlet:

1. Open a HttpConnection for the URL for which the pipeline-match is valid (will execute the pipeline, mentioned above) 2. Create a DataInputStream , using the HttpConnection object from Step 1 3. Read the XML, produced by the server , using a read-method of the DataInputStream object in a byte-array 4. Create a parser object, using the byte-array.

Putting it all together, in the Midlet, you should create a method like this:

```
  private XmlParser getParser(String url) {
    byte[] xmlByteArray = new byte[2048]; // Assume this size as the max
stream size we can receive
    XmlParser parser = null;

    // Get the XML data as byte array:
    try {
      conn = (HttpConnection)Connector.open(url); // Request the servlet
      dis = new DataInputStream(conn.openDataInputStream());
      dis.read(xmlByteArray); // Read the byte array NOW

      // Create a parser to process the xml tags:
      ByteArrayInputStream xmlStream = new
ByteArrayInputStream(xmlByteArray); // Create the input stream
      InputStreamReader xmlReader = new InputStreamReader(xmlStream); //
Give it to a stream reader
      parser = new XmlParser(xmlReader); // Create a parser
    } catch (IOException ex) {
      return null;
    }
    return parser;
  }
```

The URL points to the pipeline at the server, which will provide the data in XML.format.

With the parser, you can parse the contents. Here is an example:

```
   protected MiArticle getArticleFromXML(String barcode) {
    MiArticle art = new MiArticle();
    String url = this.baseURLXML + this.articleByBarcodeXML +
"?Barcode=" + barcode; // Assemble URL
    String tag = "";

    try {

      // Get results and create parser
      XmlParser parser = this.getParser(url); // Here I call the above
method

      // Parse the xml to get the article data:
      boolean endFlag = false; // Reset end read flag
      do {
        ParseEvent event = parser.read(); // Read a tag
        ParseEvent pe; // Declare a second parse event to read the contents
        switch (event.getType()) {

          case Xml.START_TAG:
            tag = event.getName(); // To visualize...
            if (tag.compareTo("Codi") == 0) { // Codi found
              pe = parser.read();
              art.setCodi(Integer.parseInt(pe.getText()));
            }
            if (tag.compareTo("Barcode") == 0) { // Barcode found
              pe = parser.read();
              art.setBarcode(pe.getText());
            }
            if (tag.compareTo("Nom") == 0) { // Name found
              pe = parser.read();
              art.setNom(pe.getText());
            }
            break;

          case Xml.END_DOCUMENT:
            endFlag = true; // Mark exit
            break;

          default:

        } // End switch
      } while (!endFlag);

    } catch (IOException ex) {
      System.out.println("IOException. Details: [" + ex.toString() + "].");
    } catch (Exception ex) {
      System.out.println("Exception. Details: [" + ex.toString() + "].");
    }

    if (art.hasValidData()) {
      return art;
    } else {
      return null;
    }

  }
```

As you see, I am filling in the blanks of an object of class MiArticle, using tags "Codi", "Barcode" and "Nom" in the XML, received from the server. After this action, you can display the data at the mobile's screen.

In this way, Cocoon serves data from J2EE / RMI via XML request/response to J2ME MIDP at a mobile phone...

Some observations:

- Works with Sony/Ericsson P800. Is **not** slower, compared with the "classic" way, using writeUTF() / readUTF()
- Works with PALM Tungsten, with the P800 as modem. Takes a little more time than the classic way, but this is acceptable.
- Other devices under investigation.

Resources

- The "clue" to the solution was found in the article at http://www-106.ibm.com/developerworks/library/wi-parsexml/
- For more info on how to develop under J2ME, take a look at

http://java.sun.com/j2me/download.html and the site http://developer.sonyericsson.com/site/global/newsandevents/latestnews/newsnov03/p_news114.jsp

- For P800 and other mobile phones you can use use the J2ME SDK version 1.1 of Sony/Ericsson in Borland JBuilder 9