Oracle Forms 10*g* – Java Integration Possibilities

An Oracle Forms Community White Paper

Jacob Madsen
jom@brandsoft.dk
August 2008

ORACLE

Oracle Forms 10*g* – Java Integration Possibilities

INTRODUCTION

Oracle Forms application have traditionally behaved as, and looked like, desktop applications. Back with Forms 6i, Forms consisted of a client/server architecture with lots of options to expand the functionality using ActiveX/OCX components.

However, with Forms 9i and upwards, Oracle decided to desupport the client/server architecture of Forms, thereby also desupporting the option of using ActiveX/OCX components. This meant, that a lot of these integrations to external components would no longer work with WebForms.

Luckily, with WebForms, a new world also opened with the possibilities of expanding the functionality of Forms using Pluggable Java Components (PJC's). This paper will cover the author's own experience regarding migrating a Forms 6i client/server application to Forms 10g WebForms using the integration options provided by Java.

A NOTE REGARDING JINITIATOR AND THE JAVA PLUGIN

In earlier versions of WebForms, Oracle has bundled their own custom-made version of the Java plugin called JInitiator

However, in the time of this writing, I would strongly recommend against using JInitiator and switch to JRE 1.5.0_12 or newer, alternatively 1.6.0, once it becomes certified. The reason for this is, that even if JInitiator is OK for basic Forms execution, it ties you up to the now outdated Java 1.3 runtime, which makes it impossible to utilize some very useful libraries found in newer versions of the JRE, which will be described later in this document.

Oracle themselves also now recommend people to move away from JInitiator, because it is not and will not be certified for Windows Vista, and Oracle now has a close cooperation with Sun Microsystems to work out the issues in JRE 1.5 and newer, there might still be when using the Sun JRE with Oracle Forms. See the Client Platform Statement of Direction for details:

http://www.oracle.com/technology/products/forms/htdocs/10gR2/clientsod_forms10gR2.html

However, migration to any JRE 1.6.0 should not be done without intensive tests. 1.5 of Java, from update 12 and newer, is in our experience still at this moment (July 2008) the most reliable JRE available, and we cannot recommend 1.6.0, until Sun releases their rewritten 1.6.0_10 plugin, which in this moment is in Release Candidate state, and is expected to resolve many of the issues, which have been encountered with the 1.6.0 plugins so far. For the time being, we force our users to run on JRE 1.5.0_12, but since all new machines today comes bundled with 1.6.0, we are eagerly awaiting Sun to release the final version of the 1.6.0_10 plugin, which we have been betatesting for a long time. For more information, see Sun's site about the new plugin here:

https://jdk6.dev.java.net/6u10ea.html

MIGRATION CHALLENGE 1: ORACLE REPORTS AND CLIENT INTEGRATION.

In the client/server system, there was a quite advanced printing system on top of Reports, where the user, for each RDF file, could specify either to send the report

to the Reports previewer or send it directly to a named printer, even specifying what printer tray to send specific pages to, using Reports triggers and built-ins.

For those, who do not know about the Reports 6i Engine, which you could easily integrate with Forms 6i client/server, it should be mentioned, that this engine had a lot of functionality, which per default is completely absent in WebForms. In Forms 6i, you could specify getting the report viewed in a previewer or directly to the printer, and you also had the option to send some pages to one tray at the printer, other pages to other trays etc. using built-ins provided by Reports.

How do you migrate such functionality to WebForms? Oracle themselves provides you with 2 options:

You can either let the Reports Server send the report directly to the printer – which of course in an Internet deployment is impossible, since the Reports Server of course cannot access customers' printers directly due to the 3-tier architecture.

The second option: preview a generated a PDF using a PDF reader like Adobe Reader using Web.Show_document. However, there are lots of problems with this approach too: the Adobe Reader has over time become a big and clunky piece of software, about 20 MB in download size, slow to start each time, especially on slightly older systems – forcing the user to watch a fancy splash screen on each startup – and nearly impossible to integrate with in any way, you can just print the file to the default printer using a command line, can't specify printer name, page ranges, printer trays, and other adjustments, and even to do this, you have to know the path to the Adobe Reader executable and wait for it to load each time. Oh horror…

Then what can you do about this? An obvious choice was to start re-thinking this printing system completely and begin looking at, what Java could offer in respect to print PDF's. There are 2 parts of this challenge: fetch and load the PDF from the Reports in a quick and smooth way and display it inside Forms, and a way to directly send it to a printer using a set of predefined parameters. Rendering a PDF isn't an easy task, and since Java has no direct built-in support for PDF, we had to look at 3rd party Java components to accomplish this part of the task. Fortunately, there are quite a few commercial Java components to integrate PDF support in your own application.

After doing some research, the commercial component ICEPDF from IceSoft (www.icesoft.com) was chosen, since this component seemed to fulfill the requirements when testing a wide range of PDFs generated by Oracle Reports in the system in question.

There is also a free open-source PDF Java component called PDFBox (www.pdfbox.org), which deserves an honorable mention here. Unfortunately, this component was unable to render PDF's from some of our most critical reports correctly (and still is per February 2008), so we had to go for the commercial component.

ICEPDF is a quite reliable PDF renderer, and integrating it into WebForms was relatively easy. But what about the direct printing, and our general printing system described earlier? Here we have the Java Print Service (javax.print) library introduced in JDK 1.4. Before Java 1.4, printing support in Java was really bad – it was possible, but frankly didn't work very well. The Java Print Service gave access to a lot of useful stuff all of a sudden – printer selection, page range selection, printer tray management, and since this library worked very well together with ICEPDF, it was suddenly possible to create a full-featured print system PJC to integrate in Forms together with PDFs from Reports, very similar to the previous client/server based system, providing functionality completely absent in the standard Forms/Reports 10g bundle.

Below is shown a simple demonstration of the component showing a simple report output.

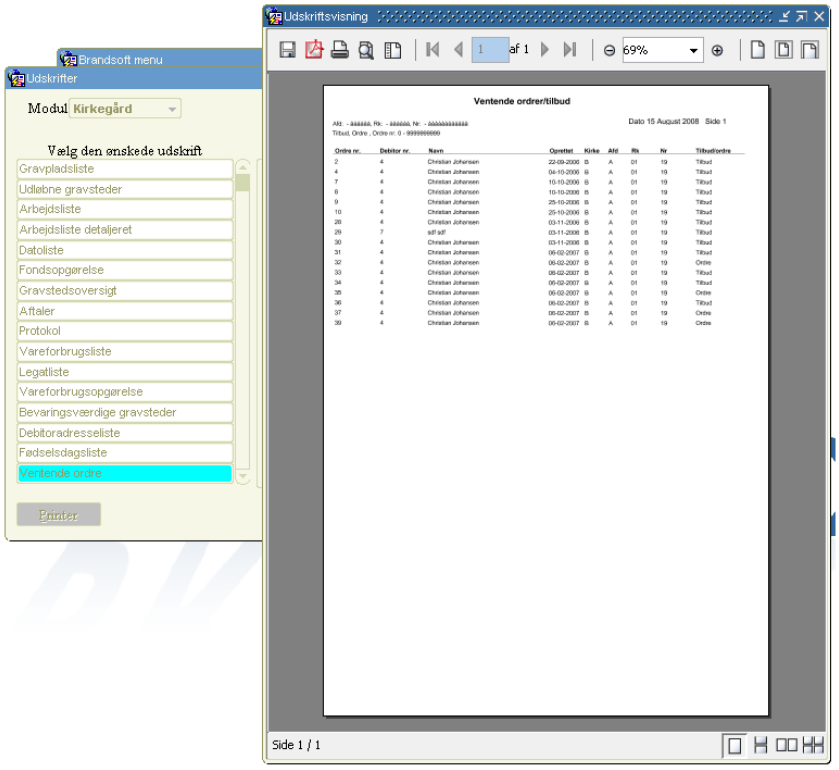

Figure 1 – the PDF viewer PJC in action in WebForms

MIGRATION CHALLENGE 2: DIGITAL MAPS INTEGRATION

Another big migration headache was a central component in the old client/server based system. The application featured digital map integration using an ActiveX component from the GIS company MapInfo, which previously had made it possible to display digital spatial map data directly in the Forms 6i-based application.

To be more specific, one of our primary niche areas for our Forms-based systems involves administration of cemeteries. To be able to manage a cemetery visually, the Forms 6i client/server system had integrated an ActiveX component from MapInfo to be able to view spatial data in MapInfo's own format, directly in Forms, such as making a selection of graves and displaying these on the map.

Unfortunately, MapInfo does not provide any good way of migrating this technology to the web. A web-based solution from MapInfo does exist, but licensing costs quickly made this path very unattractive.

How to convert the old MapInfo-based system from Forms 6i to the web then? It seemed, when moving to the web, it was time to completely rethink this part of our system too. Naturally in a web system, the map data should no more reside on each local web client, but instead be served as a web service in some way.

In this regard, I had heard about Oracle Spatial, the storage system for spatial map data, which is integrated into Oracle8i Database and newer, but this would only solve the storage part, not the visual part.

After searching a bit, I found out that Oracle actually also has a tool of their own for visually displaying data from Oracle Spatial. It's called Oracle MapViewer and consists of 2 parts: an OC4J web service to deploy in an OracleAS instance, and a client library written in Java to make it easy to integrate MapViewer with a Java-based client application or applet. For those familiar with XML, it's also possible to code an interface for the MapViewer web service yourself, but the Java client library makes this a lot easier.

This should be good – a Java client library… a PJC for WebForms should be possible. This Java component just requires JRE 1.5 or newer, and at the beginning of this project, there was still some issues regarding the 1.5 Java plugin and WebForms. But these issues fortunately was worked out, and when 1.5.0 update 12 was finally released by Sun – and certified for use in Oracle Applications even – I was confident enough, that it would be safe to upgrade our existing 1.4.2 plugin, which the customers used that that time – finally.

A sample screenshot from the current version of my MapViewer PJC can be seen below. As you might notice, the interface surrounding the digital map is clearly Forms, but the map itself is fetched from the MapViewer server using the MapViewer client library.

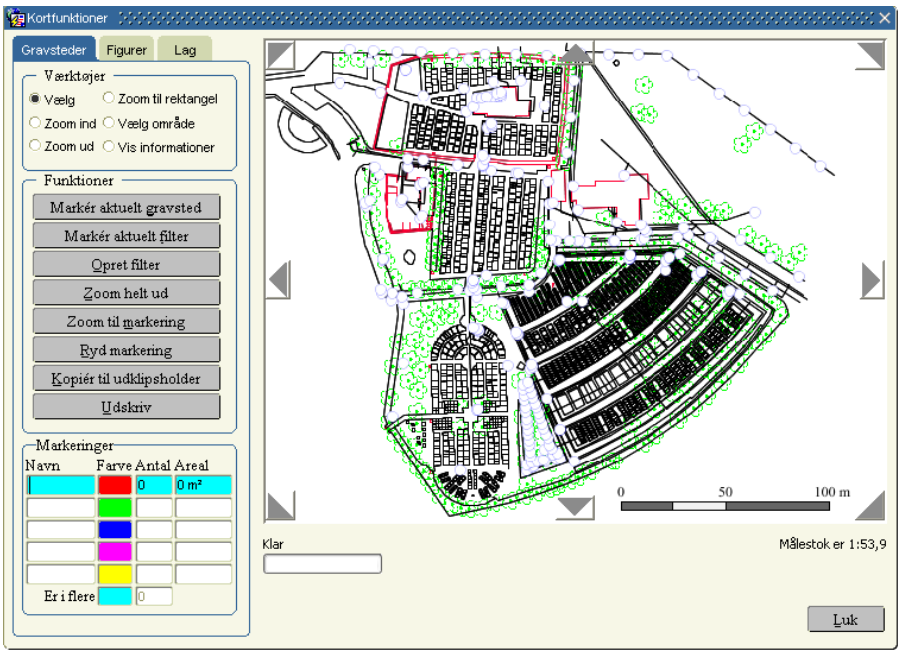

Figure 2 – the Mapviewer client library PJC in action in Webforms

With this Java component and Oracle Spatial+MapViewer, it was even possible to integrate the map system a lot better, than was previously possible – Forms can query the Spatial tables in the database directly for misc. information, and the PJC mechanisms make it possible to send data back and forth between Forms and MapViewer rather easily.

However, it must be said, that the implementation of this PJC was not exactly done overnight and required extensive testing, but the end result has proved to be very satisfying for the users – the response from them has been very positive.

How to convert the actual data from the MapInfo format to Oracle Spatial was another big part of this project, but I believe going into details regarding this will be going beyond the scope of this document, which only is concerned about the Forms visual map front-end.

SUMMARY

The solutions described in this paper is intended to serve as a demonstration and inspiration for the possibilities of greatly expanding the functionality of Oracle Forms 10g using Pluggable Java Components (PJC).

ABOUT THE AUTHOR:

Jacob Madsen works as developer for a Danish independent software vendor, whose main business has become hosting self-developed Forms application on Oracle Application Server 10g. He has worked with Oracle technologies, especially Forms and Reports, since 2000, where he got a headstart into the Oracle world by co-writing a graduation essay about a migration of a Forms 6i client/server application to web-based Forms (not the system mentioned in this paper). For questions and further information regarding the examples mentioned in this paper, please contact Jacob at jom@brandsoft.dk.

Oracle Forms 10*g* – Java Integration Possibilties
August 2008
Contributing Authors: Frank Nimphius, Grant Ronald (content review)

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com