Integrating Crystal Reports with a J2EE Application: A Case Study

Introduction

In this paper, we discuss our experience with integrating Crystal Reports with a J2EE application, Task Workbench (TWB). The goal of this paper is to help others evaluate the benefits and drawbacks of using Crystal Reports in the context of a J2EE application.

Task Workbench (www.taskworkbench.com) is a work management tool that provides real-time information about projects and day-to-day operations. It is a web application that uses JSPs for the presentation layer and EJBs for the business logic. Persistence is via container managed persistence (CMP) Entity Enterprise Java Beans (EJBs) mapped to an Oracle relational database.

TWB helps project managers stay informed about the progress of projects by providing various ways to track the number of hours that employees spend on each project. When designing the this application, an important requirement was to provide project managers and administrative staff with a high-level view of the team member and project data.

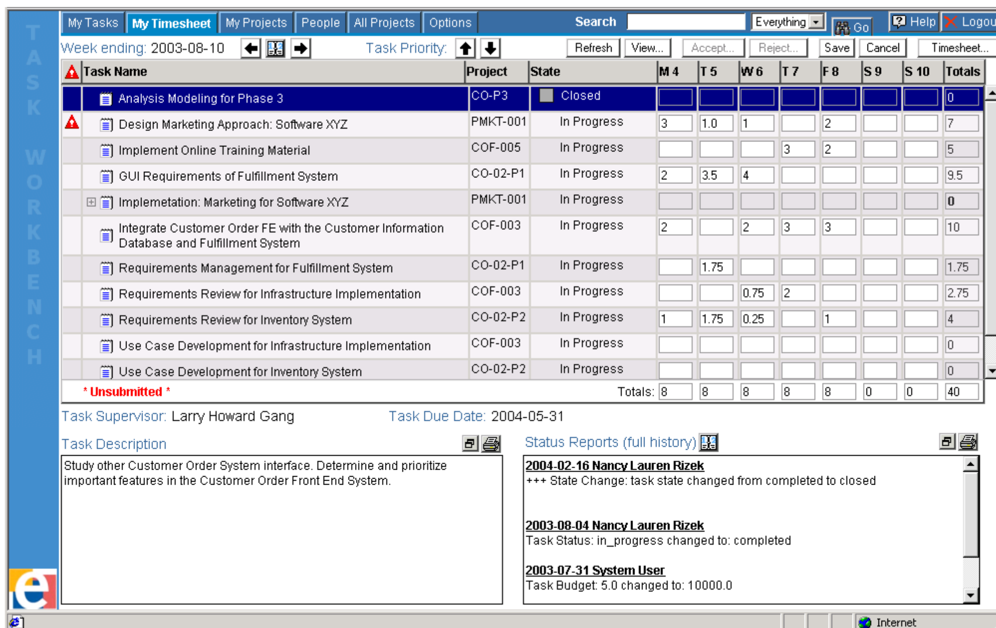

Figure 1: Task Workbench displays an employee's tasks and the hours spent on each task.

Initially we created reports in TWB using JSP, however this approach proved to be prohibitively time consuming, especially for complex reports, so a Java reporting solution for web applications was sought. Our basic requirements were that the solution should make it fast and easy to create and modify reports, and that it could be integrated with a J2EE application with minimum implementation effort.

We selected the Crystal Reports Java Reporting Component because it met all of our requirements. It is also a mature, industry-leading product with a range of tools available for it. This was an important consideration because we wanted to ensure that our reporting solution could grow as TWB's requirements expanded.

The first section of this paper briefly describes the original report implementation using hand-coded JSP, and the problems that we encountered with this approach. The next section explains why we selected the Crystal Java Reporting Component as our reporting solution. The remainder of the paper discusses how we integrated Crystal Reports with TWB, and details the pros and cons of using Crystal Reports.

Original Report Implementation

The early versions of TWB implemented reports using hand-coded JSP. In most cases, the report data used JDBC to connect directly to the database for performance reasons. An additional advantage of implementing reports in this way was that the queries already existed for a precursor application, and these complex queries had been proved to be correct.

This approach quickly became problematic for a number of reasons:

- Query optimization was difficult since the queries had to be dynamically altered, based on multiple report settings.
- The implementation effort for reports with grouped data proved to be prohibitive.
- We had no practical way to rapidly prototype reports for cases where the requirements were unclear or unstable. This discouraged experimentation and lead to certain reports not adequately meeting the customer's requirements.
- Maintenance became very time-consuming as the schema changed and new features were introduced.
- Customization of the output of the reports for specific uses (e.g. printing) was time-consuming.

Our initial implementation used CMP entity EJBs to obtain the data for reports. The shortcomings of this naïve implementation quickly become apparent – in particular, it did not scale well for large datasets and a significant amount of work would have been required to address this deficiency.

These issues made it clear that we needed a much better solution to our reporting requirements; we needed a way to create more sophisticated reports with much less effort. We turned to Crystal Reports for this solution.

Selecting a Reporting Solution

Our difficulties with creating reports with JSP in TWB caused us to look to better reporting solutions. Our requirements were that the reporting solution should:

- Have a visual report editor to make it fast and easy to create and modify reports.
- Generate highly presentable reports, both on the screen and when printed.
- Include the ability to generate grouped reports with graphs.
- Obtain the report data via an instance of javax.sql.DataSource, or a JDBC connection.
- Be a pure Java solution.

We considered open source alternatives such as Jasper Reports. However, we found that none of the open source options met our basic requirement for a visual report editor and we judged them unlikely to provide adequately sophisticated solutions without a lot of additional work. We elected to use a solution based on Crystal Reports, because we had been impressed with Crystal Reports when we used it with an unrelated project.

Copyright 2003, Ensemble Systems Inc.

There are three classes of Crystal Reporting solutions: Java Reporting Component, Report Application Server (RAS), and Crystal Enterprise. The Java Reporting Component provides a simple, pure Java solution for viewing reports in cases where there is likely to be low concurrent usage and where report management functionality is not required. The RAS provides a basic set of services for processing, viewing and modifying reports locally and remotely. Since report processing is done on demand, RAS is best suited for smaller datasets and small numbers of concurrent users. Crystal Enterprise is a set of products with features designed to support large installations with more demanding requirements. The basic features of Crystal Enterprise include load management (clustering, report scheduling, etc), availability (fail-over), and row/column level security.

We selected the Crystal Java Reporting Component for TWB since this solution met all our requirements and we anticipated that there would be a small number of concurrent users for reports. We also did not need the advanced report interactivity options as users would only require basic report viewing and printing capabilities.

Implementation

The Crystal Java Reporting Component was integrated into TWB with emphasis on two key goals: The first was to provide an implementation that required the least possible effort to add new reports and maintain existing reports. The second goal was to make it simple to switch to a more sophisticated Crystal reporting solution, such as Crystal Enterprise, in order to provide for scalability if required.

The Crystal Java Reporting Component has a JSP tag library that provides a simple way to display basic reports; direct access to the API is provided for cases where more features are required. We chose to implement our own custom JSP tag that uses the Crystal Reporting Component to display the reports. This approach allowed us to customize the appearance of the reports and to set report parameters while keeping the design simple and easy to maintain.

The simplicity of this approach is illustrated by the fact that the only Java code required in the JSP is a single method call to a factory class to obtain an instance of the report to be displayed on the page. Figure 2, simple_banked_time_report.jsp provides a basic example of a JSP that might be used to request a report. This form passes the required parameters to view_report.jsp (Figure 3).

Copyright 2003, Ensemble Systems Inc.

www.ensemble-systems.com

```
<%@ page import="com.ensemsys.twb.presentation.crystal.ReportParameter,
                 com.ensemsys.twb.presentation.crystal.ReportType"%><html>
<head>
  <title>Report Viewer</title>
  <link rel="stylesheet" href="style/main.css" type="text/css"/>
</head>

<body>

  <h1>Simple Banked Hours Report</h1>
  <p>Enter the date range for the report and click &quot;Submit&quot; to view
   the report.</p>

  <form method="post" action="view_report.jsp">
    <input type="hidden" name="<%=ReportParameter.TYPE%>"
      value="<%=ReportType.BANKED_TIME_SIMPLE%>"/>
    <table>
    <tr>
      <td class="label">Start Date:</td>
      <td>
        <input type="text" name="<%=ReportParameter.START_DATE%>"
          value="2003.06.30"/>
      </td>
    </tr>
    <tr>
      <td class="label">End Date:</td>
      <td>
        <input type="text" name="<%=ReportParameter.END_DATE%>"
          value="2003.12.01"/>
      </td>
    </tr>
    <tr>
      <td> </td>
      <td style="text-align: right">
        <input type="submit" name="submitBtn" value="Submit"/>
      </td>
    </tr>
    </table>
  </form>

</body>
</html>
```

Figure 2: simple_banked_time_report.jsp, an example of a simple form that can be used to display a report.

Copyright 2003, Ensemble Systems Inc.
www.ensemble-systems.com

```
<%@ page import="com.ensemsys.twb.presentation.crystal.ReportFactory,
                 com.ensemsys.twb.presentation.crystal.Report"%>
<%@ taglib uri="/simplereportviewer.tld" prefix="viewer" %>

<html>
<head>
  <title>Report</title>
  <link rel="stylesheet" href="style/main.css" type="text/css"/>
</head>
<body>

<%
  // "CrystalEventTarget" is the report action name used by Crystal
  final String REPORT_NAME = "CrystalEventTarget";
  String reportName = (String) session.getAttribute( REPORT_NAME );

  // refresh might have done via a report action
  if ( reportName == null )
  {
    reportName = (String) request.getParameter( REPORT_NAME );
  }

  Report report = null;

  if ( reportName == null )
  {
    report = ReportFactory.newInstance( request );
    reportName = report.getName();
    session.setAttribute( REPORT_NAME, reportName );
    session.setAttribute( reportName, report );
  }
  else
  {
    report = (Report) session.getAttribute( reportName );
  }

%>
<viewer:viewer report="<%=report%>" />

</body>
</html>
```

Figure 3: view_report.jsp, the JSP used to display a report.

The class ReportFactory creates an instance of Report, using the parameters in the request to determine the type of report to be created and to create the report's parameters (see Figure 4). This approach neatly encapsulates the business logic specific to individual reports in one method. Factory methods for more complex types of reports are best implemented in a subclass of ReportFactory - the ReportFactory can then delegate to the subclass when creating that type of report.

Copyright 2003, Ensemble Systems Inc.
www.ensemble-systems.com

```
package com.ensemsys.twb.presentation.crystal;

import com.crystaldecisions.sdk.occa.report.data.Fields;
import com.crystaldecisions.sdk.occa.report.data.ParameterField;
import com.crystaldecisions.sdk.occa.report.data.ParameterFieldDiscreteValue;
import com.crystaldecisions.sdk.occa.report.data.Values;

import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import java.sql.Date;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;


/**
 * Provides methods for creating an instance of a Report
 * from the parameters in the request.
 */
public class ReportFactory
{
  public static final DateFormat FORMAT =
      new SimpleDateFormat( "yyyy.MM.dd" );


  public static Report newInstance( ServletRequest request )
    throws JspException
  {
    String reportType = getParameter( request, ReportParameter.TYPE );
    Report report = null;

    // For brevity, only the simple report type is shown here
    if ( reportType.equals( ReportType.BANKED_TIME_SIMPLE ) )
    {
      report = newSimpleBankedTimeReport( request );
    }
    else
    {
      throw new InvalidReportTypeException( reportType );
    }

    return report;
  }
```

Figure 4: ReportFactory.java creates an instance of a Report, based on parameters submitted from a form.

Copyright 2003, Ensemble Systems Inc.
www.ensemble-systems.com

```
  private static Report newSimpleBankedTimeReport( ServletRequest request )
    throws JspException
  {
    Date startDate = getDateParameter( request, ReportParameter.START_DATE );
    Date endDate = getDateParameter( request, ReportParameter.END_DATE );

    Fields fields = new Fields();
    ParameterField prevYearField = newDateField( "Prev Fiscal Year", startDate
);
    ParameterField reportDateField = newDateField( "Report Date", endDate );
    fields.add( prevYearField );
    fields.add( reportDateField );

    return new Report( "protected/reports/bhr_simple_demo.rpt",
      ReportType.BANKED_TIME_SIMPLE, fields );
  }


  public static ParameterField newDateField( String name, Date date )
  {
    ParameterField field = new ParameterField();
    Values vals = new Values();
    ParameterFieldDiscreteValue value = new ParameterFieldDiscreteValue();
    field.setName( name );
    value.setValue( date );
    field.setReportName( "" );
    value.setDescription( "" );
    vals.add( value );
    field.setCurrentValues( vals );
    return field;
  }


  public static Date newDate( String s )
    throws InvalidDateStringException
  {
    java.sql.Date date = null;

    try
    {
      date = new java.sql.Date( FORMAT.parse( s ).getTime() );
    }
    catch ( ParseException e )
    {
      throw new InvalidDateStringException( e );
    }

    return date;
  }


  protected static void assertHasParameter( ServletRequest request, String name
)
  {
    String parameter = request.getParameter( name );

    if ( parameter == null || parameter.equals( "" ) )
    {
      throw new NoSuchParameterException( name );
    }
  }
```

Figure 4: ReportFactory.java (continued).

Copyright 2003, Ensemble Systems Inc.
www.ensemble-systems.com

```
  protected static Date getDateParameter( ServletRequest request, String name )
    throws JspException
  {
    String parameter = getParameter( request, name );
    Date date = null;

    try
    {
      date = newDate( parameter );
    }
    catch ( InvalidDateStringException e )
    {
      throw new JspException( e );
    }

    return date;
  }


  protected static String getParameter( ServletRequest request, String name )
  {
    assertHasParameter( request, name );
    return request.getParameter( name );
  }
}
```

Figure 4: ReportFactory.java (continued).

The class SimpleReportViewerTag (Figure 5) renders the report; this class uses the Crystal Java Reporting Component API to perform the drawing of the report. This class is quite straightforward - it creates an instance of CrystalReportViewer, sets the database connection information and the parameters that the report requires, and then displays the report. The corresponding TLD, simplereportviewer.tld, is shown in Figure 6.

This approach is very clean and is easy to maintain, as the design ensures that the logic for each report is in a single location.

```
package com.ensemsys.twb.presentation.crystal;

import com.crystaldecisions.report.web.viewer.CrystalReportViewer;
import
com.crystaldecisions.reports.reportengineinterface.JPEReportSourceFactory;
import com.crystaldecisions.sdk.occa.report.data.ConnectionInfo;
import com.crystaldecisions.sdk.occa.report.data.ConnectionInfos;
import com.crystaldecisions.sdk.occa.report.data.Fields;
import com.crystaldecisions.sdk.occa.report.data.IConnectionInfo;
import com.crystaldecisions.sdk.occa.report.lib.ReportSDKExceptionBase;
import com.crystaldecisions.sdk.occa.report.reportsource.IReportSource;
import com.crystaldecisions.sdk.occa.report.reportsource.IReportSourceFactory2;
import com.ensemsys.twb.ApplicationProperties;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
import java.util.Locale;
```

Figure 5: SimpleReportViewerTag.java is the implementation of the custom JSP tag for displaying a Report.

Copyright 2003, Ensemble Systems Inc.

```
/**
 * Displays a Crystal report.
 */
public class SimpleReportViewerTag extends TagSupport
{
  private Report report;


  public Report getReport()
  {
    return report;
  }


  public void setReport( Report report )
  {
    this.report = report;
  }


  public int doEndTag() throws JspException
  {
    CrystalReportViewer viewer = newViewer( report );

    try
    {
      IReportSourceFactory2 rptSrcFactory = new JPEReportSourceFactory();
      IReportSource reportSource =
          (IReportSource) rptSrcFactory.createReportSource(
            report.getReportFileName(), getLocale() );

      viewer.setReportSource( reportSource );
      viewer.setDatabaseLogonInfos( newDBConnectionInfos() );
      viewer.setEnableLogonPrompt( false );

      Fields fields = report.getFields();
      if ( fields != null )
      {
        viewer.setParameterFields( fields );
        viewer.setEnableParameterPrompt( false );
      }

      viewer.processHttpRequest(
          (HttpServletRequest) pageContext.getRequest(),
          (HttpServletResponse) pageContext.getResponse(),
          pageContext.getServletConfig().getServletContext(),
          pageContext.getOut() );
    }
    catch ( ReportSDKExceptionBase e )
    {
      throw new JspException( e );
    }
    finally
    {
      if ( viewer != null )
      {
        viewer.dispose();
      }
    }

    return EVAL_PAGE;
  }
```

Figure 5: SimpleReportViewerTag.java (continued).

```
  private Locale getLocale()
  {
    // generally you want to change this method
    // to return the specific locale that your J2EE
    // application is using
    return pageContext.getRequest().getLocale();
  }


  private static ConnectionInfos newDBConnectionInfos()
  {
    ConnectionInfos infos = new ConnectionInfos();
    IConnectionInfo con = new ConnectionInfo();
    con.setUserName( ApplicationProperties.getJDBCUser() );
    con.setPassword( ApplicationProperties.getJDBCPassword() );
    infos.add( con );
    return infos;
  }


  private static CrystalReportViewer newViewer( Report report )
  {
    CrystalReportViewer viewer = new CrystalReportViewer();
    viewer.setName( report.getName() );

    // set the viewer formatting and behaviour options
    //viewer.setSeparatePages( false );
    //viewer.setBestFitPage( true );
    // ... and so on

    return viewer;
  }
}
```

Figure 5: SimpleReportViewerTag.java (continued).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>Crystal Viewer Tag Library</short-name>
    <description>Display a Crystal Report with paramters</description>
    <tag>
        <name>viewer</name>
        <tag-
class>com.ensemsys.twb.presentation.crystal.SimpleReportViewerTag</tag-class>
        <body-content>JSP</body-content>
        <attribute>
            <name>report</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
  </tag>
</taglib>
```

Figure 6: simplereportviewer.tld - the TLD for the SimpleReportViewer tag extension.

Since TWB only allows project managers to view reports, TWB's existing access control mechanisms were sufficient for controlling access to reports. It is anticipated that a future version of TWB may require more fine-grained access control - for example, users may be restricted to seeing data about specific projects on a report. In this case, we would utilize the Crystal Enterprise security features for this level of access control.

Evaluation of Implementation

Our report implementation, using a combination of Crystal Reports and the Crystal Java Reporting Component had several advantages and a few disadvantages when compared with our previous JSP implementation.

The biggest advantage was the speed and simplicity of creating visually appealing reports. An important factor contributing to the speed of creating reports is that the Crystal Reports visual editor makes it possible to design and edit reports without having to build and deploy TWB. An additional benefit is that the ease of creating and modifying reports means that the developer does not need to have expertise in J2EE, meaning that a larger pool of developers are available to work with the reports.

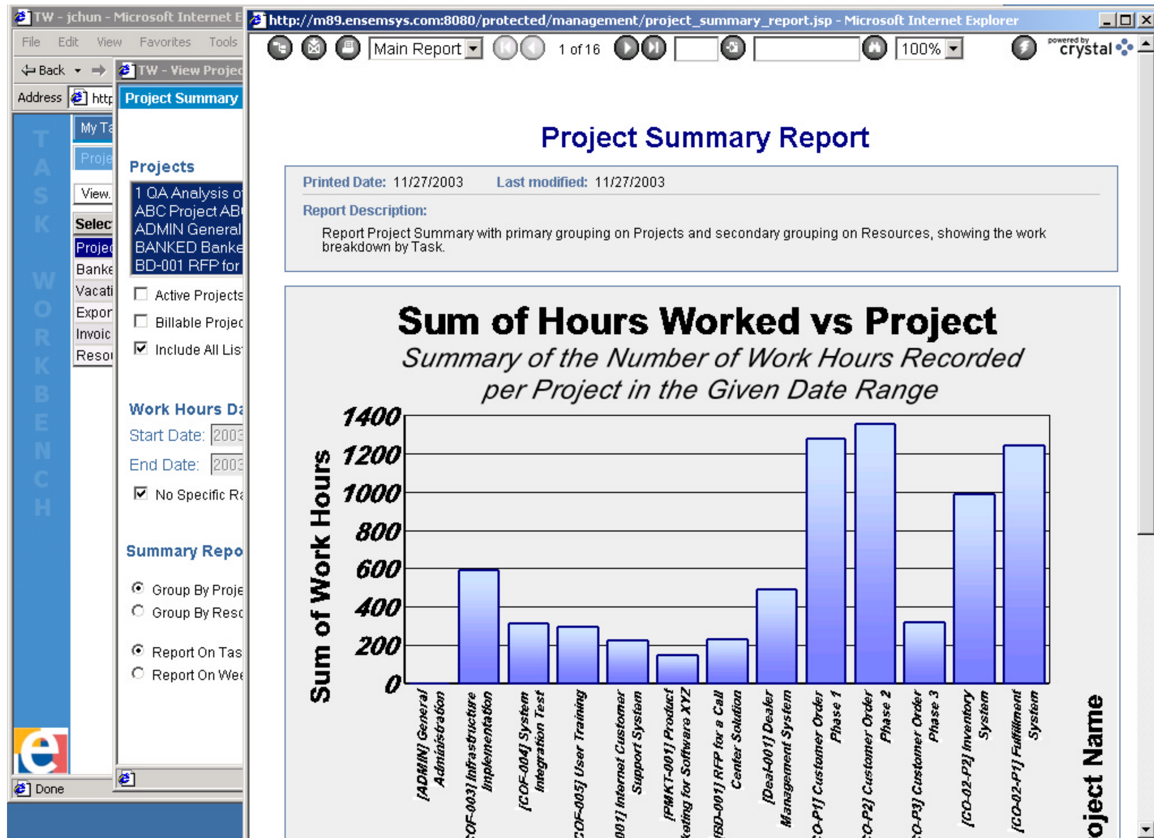

Figure 7: Example of a Crystal Report integrated with Task Workbench.

The Crystal API made it very simple to integrate the reports into a web application. Using a custom JSP tag took this simplicity a step further, with very little code being required to display a report. Making changes to reports is now almost trivial. Huge time savings were also

realized by the fact that we did not need to write code to support requirements such as printing, exporting reports, tuning queries, etc.

A comparison of the implementation effort of the original approach using hand-coded JSP versus the equivalent implementation with Crystal Reports is shown in Table 1.

As outlined in this document, one of the reasons we choose Crystal technology is that Crystal Decisions (recently acquired by Business Objects) offers a number of solutions to address scalability issues, particularly in the Crystal Enterprise suite. The performance experienced when using the Crystal Java Reporting Component was adequate for our current requirements, but under higher loads we will need to switch to a more full-featured Crystal solution. This is a very simple change, due to the way we have integrated Crystal Reports with TWB. Essentially, the only changes required would be to alter the implementation of the SimpleReportViewerTag to use a Crystal RAS (or Crystal Enterprise) report source, and to alter the path to the reports themselves, in the ReportFactory class.

Task	Original JSP	Crystal Reports
One-time Setup/Familiarization (Assumes JSP/java knowledge)	3 days (JDBC, queries)	4 days (Product, Integration)
Initial Report Design and Implementation	4 days (input and output pages) 10 days (creating and optimizing queries, organizing the data returned to match the output format)	2 days (input pages-leveraged existing) 1 day (Crystal Reports Designer)
Make reports available as CSV	8 days (summary, create hierarchy CSVs)	1 day
Integration/Test/Maintenance	5 days (introduction of sub-projects, project groups)	1 day
Total	30 days	9 days

Table 1: Comparison of implementation effort in TWB for hand coded JSP -Vs- Crystal Reports.

The main disadvantage of using Crystal Reports in TWB is that some reports are not easy to re-use across different databases. Naturally, our development and QA environments use different databases from the production environment, and it is important to be able to deploy the application using an arbitrary database type/schema that have the same tables but are otherwise different. The only solution available to us has been to create reports by hand for each database.

Crystal Reports connect to the database directly using SQL via JDBC. This allows reports to perform better, but the trade-off is that the report is dependent on a database schema that is automatically generated and updated when the CMP entity beans are built. The developers responsible for the CMP entity beans need to keep this in mind when they make any changes. Using the database directly in the context of an EJB application can lead to concurrency issues, so care must be taken to ensure that the EJB container is configured for non-exclusive access.

Despite the drawbacks mentioned, using Crystal Reports has proved to be a much better approach to creating reports for TWB, because of the huge time savings at both, the

Copyright 2003, Ensemble Systems Inc.

implementation and maintenance phases of development. In short, integrating Crystal Reports with TWB has allowed us to produce a superior reporting capability with room for future requirement expansion - in a fraction of the time that it took to implement a hand-coded JSP solution.

Summary

The Crystal Java Reporting Component successfully fulfilled our requirements for a sophisticated Java reporting solution that was simple to integrate with a J2EE application and that provides high quality reports which are easy to create. Our method of integration, using a custom JSP tag combined with a factory class, provided a very simple way to integrate Crystal Reports with TWB. The main drawback of using Crystal Reports is the lack of portability of individual reports, but the magnitude of the productivity gains we experienced using Crystal Reports far outweighed this.

Copyright 2003, Ensemble Systems Inc.
www.ensemble-systems.com