Paper 010-31

Hiding the Complexity: A Customized Application for Accessing Multidimensional Data

Phil Rhodes, Baylor University, Waco, TX

ABSTRACT

The mission of the Office of Institutional Research and Testing (IRT) at Baylor University is to conduct research in order to provide information which supports institutional planning, policy information, and decision making. As part of that mission, IRT maintains several data warehouses with student, financial, and human resource data. Warehouse users access the data with a variety of tools, including a customized MDDB report viewer built using JavaServer Pages and the Java classes provided with webAF. This report viewer allows users to lay out reports, export the data to a spreadsheet, and save reports for later use. This paper will describe the design and building of the report viewer. Additionally, a separate application will be described that allows the users to pre-filter the data before viewing the report, thus hiding some of the complexity of the underlying data.

SAS products used include Base SAS®, SAS/MDDB® Server, and the webAF™ Java classes from SAS® AppDev Studio™ (version 2).

INTRODUCTION

The mission of the Office of Institutional Research and Testing (IRT) at Baylor University is to conduct research in order to provide information which supports institutional planning, policy information, and decision making. As part of that mission, IRT maintains three different data warehouses (running on SAS software) with student, human resources, and financial data. Access to this data is provided online via a web browser.

The online access to the student warehouse was initially provided using Microsoft's Active Server Pages, the SAS/Intrnet application dispatcher, and the SCL-based MDDB Report Viewer included with SAS/Intrnet®. The current version uses the SAS Public Sector Group Portal, the application dispatcher, SAS® Integration Technologies, and a customized MDDB report viewer built using the webAF Java classes, as well as static reports. This paper will focus on the design and construction of the customized MDDB report viewer and a separate application that allows users to pre-filter data to simplify the report.

THE CUSTOMIZED MDDB REPORT VIEWER

The customized MDDB report viewer (MRV) is a replacement for the old SCL-based MDDB Report Viewer shipped with SAS/Intrnet. Written in Java and using JavaServer Pages, the MRV is based on the example given in the JavaDocs for the webAF class `com.sas.servlet.beans.mddbtable.html.MDTable` (see References). This example provides a working report viewer accessing the SASHELP.PRDMDDB data source. However, it does not include many features a full reporting application needs: ways to customize the display, save reports, export data, access controls, etc. Additionally, the data source and initial report layout were hardcoded, which was not an option for a general report viewer.

ADDING A MENU BAR

The first step in creating a customized MRV was to add a menu bar to allow the user to modify the report layout, filter data, and add totals. Additionally, users needed to be able to export the data to a spreadsheet and save report layouts for later use. IRT also wanted to provide direct access to help and documentation and to a data dictionary.

To create a menu bar requires three steps:

- Create menu items (for MDDB commands only)
- Create menus for MDDB commands and simple links
- Add the menu items to menus
- Add the menus to a menu bar

The `MenuItem`, `Menu`, and `MenuBar` classes are part of the `com.sas.servlet.beans.html` package (see References).

Basic menus with simple links are created using a label for the item, an image, and a URL. The following code shows the creation of menus to access the data dictionary and save a report:

```
Menu dictionaryMenu  = new Menu("Dictionary","../graphics/Bookmark.gif",
                                application.getInitParameter("DataDictionaryLoc"));
dictionaryMenu.setTarget("_blank");
Menu saveMenu  = new Menu("Save","../graphics/save.gif",
    "Javascript:headerWin('"+application.getInitParameter("SaveMDDBPage")+"')");
```

Note that these are simple, one item menus which open the URL when clicked – they contain no `MenuItems`.

Menus that interact with MDDB report (to change the layout, filter data, etc.) require the creation of `MenuItem` objects representing the commands to perform. These are created using the `MDSelectorMenuItem` from `com.sas.servlet.beans.mddbtable.html`.

The following code creates a menu item to allow users to modify the report layout:

```
MDSelectorMenuItem dimSelector = new MDSelectorMenuItem();
dimSelector.setModel(mddbModel);
dimSelector.setSelectorType(MDSelectorMenuItem.QUERY_SELECTOR);
```

where `mddbModel` is the reference to the MDDB table model object. The menu items for a filter selector and totals selector are created similarly, using `MDSelectorMenuItem.SUBSET_SELECTOR` and `MDSelectorMenuItem.TOTALS_SELECTOR`.

Once the menu items were created, they are added to a menu:

```
Menu dimMenu = new Menu("Layout","query.gif");
dimMenu.add(dimSelector);
```

This creates a new Menu object with a text label of "Layout" and a graphic named "query.gif".

Enabling export to a spreadsheet requires an additional step. First, an `MDExportToExcel` object must be created. This will generate HTML that will submit a command to the Command Processor and cause the current report to be exported to Excel (in CSV format). Then a new `Menu` with a custom action is created:

```
MDExportToExcel exporter = new MDExportToExcel();
exporter.setFormName(exportForm);
exporter.setFormAction("portalMRV.jsp");    // the MRV page
exporter.setRequest(request);               // request is the JSP request object

Menu exportMenu = new Menu("Export","../graphics/menu_filexcel.gif");
exportMenu.setCustomAction("document."+exportForm+".submit();");
```

Note that the custom action uses the form name of the `MDExportToExcel` object. This sets the menu to submit the form when clicked instead of using a simple URL reference.

Finally, all of the Menus are added to a menu bar:

```
MenuBar mddbMenu = new MenuBar();
mddbMenu.add(dimMenu);         //layout
mddbMenu.add(subsetMenu);      //filter
mddbMenu.add(totalMenu);       //totals
mddbMenu.add(exportMenu);      //export
mddbMenu.add(dictionaryMenu);  //data dictionary
mddbMenu.add(saveMenu);        //save report
mddbMenu.setMenuType(MenuBar.SELECTOR_EXPAND);
mddbMenu.setRequest(request);
```

The request object is the `HttpServletRequest` object provided by the servlet container.

SETTING INITIAL REPORT LAYOUTS

In the example report viewer from SAS, the initial configuration parameters were hardcoded in the JSP. For a general report viewer, this will not work. One option is to include the metabase, database, and report layout variables as request parameters. However, this would make it difficult to determine the initial layout for a report without inspecting the raw HTML. Additionally, the links to the MRV would be long and complex, especially since many of the hierarchies used in the report layouts have spaces in them, leading to cryptic URLs such as

```
http://localhost/sasportal/mrv/portalMRV.jsp?row=Academic%20Unit&col=...
```

To avoid these problems, the initial report layouts are stored in a Java properties file. The first time an MDDB report is requested, a new `MDDBConfiguration` object named `mddbConfig` is created. This object reads in the properties file and provides the initial configuration information for each of the MDDBs. This allows one JSP to provide access to all of the MDDBs without hardcoding configuration information. Multiple values can be specified by separating them with commas:

```
ugrad.mddb_freshman_retention.label=Freshman Retention Data
ugrad.mddb_freshman_retention.column=Term
ugrad.mddb_freshman_retention.row=Unit-Dept-Program
ugrad.mddb_freshman_retention.measure=sathigh
ugrad.mddb_freshman_retention.statistic=N,MEAN
ugrad.mddb_freshman_retention.metabase=UGMETA
ugrad.mddb_freshman_retention.database=UGRAD.MDDB_FRESHMAN_RETENTION
```

The keys are of the form `<MDDB library>.<MDDB name>.<configuration parameter>` and are retrieved from `mddbConfig` when the report is being set up:

```
//Get row, column, measures, and statistics
String col[] = mddbConfig.getProperty(mddbName,"column").split(",");
String row[] = mddbConfig.getProperty(mddbName,"row").split(",");
String measure[] = mddbConfig.getProperty(mddbName,"measure").split(",");
String stat[] = mddbConfig.getProperty(mddbName,"statistic").split(",");
```

Figure 1 shows the customized MDDB Report Viewer with the menu bar and an initial report layout.

Layout Filter Totals Export Dictionary Save

Data source: Undergraduate Enrollment Data

Year		2000-2001	2001-2002	2002-2003	2003-2004	2004-2005	2005-2006
		Enrolled Count	Enrolled Count	Enrolled Count	Enrolled Count	Enrolled Count	Enrolled Count
Academic Unit		Total Number of Nonmissing Values	Total Number of Nonmissing Values	Total Number of Nonmissing Values	Total Number of Nonmissing Values	Total Number of Nonmissing Values	Total Number of Nonmissing Values
Arts & Sciences	>	12854	13799	14006	14365	14683	8654
Business	>	7911	7964	7586	6795	6168	3534
Education	>	3013	2943	2576	2170	1892	1198
Engineering & Computer Science	>	1244	1327	1232	1064	995	621
Honors College	>	.	.	389	352	369	219
Institutes & Special Studies	>	404	498	180	163	164	73
Music	>	586	622	669	615	609	338
Nursing	>	783	788	857	957	1066	559
Social Work	>	.	.	.	.	.	93

Figure 1: Customized MDDB Report Viewer

ADDITIONAL CUSTOMIZATIONS

Two additional features are also part of the final MRV. First, Cascading Style Sheets (CSS) are used to give the report viewer the same look as the rest of the Baylor University's web site. This also simplified updates as the official look changes. Second, the process of reaching through to detail data is modified. The data available to users is filtered to remove certain records. Records are filtered for two reasons: missing measure values and access control.

One of the "features" of the base MDDB data model is that it returns all of the records for a cell when detail data is requested. This is normally what the user would want. However, when the statistics shown is 'Total Number of Nonmissing Values', the model still returns all records, including those with missing measure values. One method to handle this would be to subclass the underlying MDDB data model and substitute one with appropriate behavior. However, since the detail data also needed to be subset for access control, another method is used.

For Baylor's data warehouse, the general policy has been that anyone can see summary data, but can only see detail data for students in their areas (i.e., the English department chair can only see records of English majors; the dean of the School of Business can only see students with majors in the School of Business). To provide this access control, detail data is subset prior to be displayed using the detail data model underlying the MDDB data model.

The process to subset detail data is

- Trap the show detail data command
- Get the user's access level
- Get the underlying detail data model
- Create a string containing a WHERE clause
- Apply the WHERE clause to the detail data model

To trap the show detail data command, compare the mask of the current command being processed by the command processor (`com.sas.servlet.beans.mddbtable.MDCommandProcessor`) to `com.sas.sasserver.mdtable.commands.MDCommand.SHOW_DETAIL_DATA_COMMAND`. If they are equal, the command is the show detail command.

The access level defines what detail data the user is allowed to see. It is part of the user's portal profile and is stored in the user's session.

The underlying detail data model is retrieved from the show detail command:

```
DataSetInterface dsi = (DataSetInterface)showDetail.getShowDetailTableModel();
```

Finally, the WHERE clause is created and applied to the data:

```
String whereClause = "(" + measureName + " ne .) and (" + accessVar +
                     " = " + accessLevel + ")";
dsi.setWhere(whereClause);
```

Note that the WHERE clause eliminates both records with missing measure values (measureName ne .) and any records outside of the user's access level.

PRE-FILTERING THE MDDB DATA

For the majority of the MDDBs available in the data warehouse, the initial report layout provides a good starting point. The retention data sources, however, are much more complex and the questions asked tend to be more focused on a specific academic program or term. For these MDDBs, a separate page that allows the users to make choices about what to display reduces the volume of data displayed and makes it easier for users to find answers. For example, if a user is only interested in students who entered in a particular program in a particular year, only the data for that program in that year is displayed in the report viewer for their analysis.

The key to performing this pre-filtering is understanding both the physical structure of the underlying MDDB (the "cube") as well as the data currently being displayed (the "table"). The package `com.sas.sasserver.mdtable` contains the classes and interfaces necessary for pre-subsetting the data, and the package JavaDocs (see References) have an excellent overview of the structure used.

In brief, the "cube" is made up of `Dimensions`, which contain `Hierarchies`. The `Hierarchies` contain `Levels`, and each `Level` contains `Members` which represent the values. Note that a `Hierarchy` may consist of a single

Level. Conceptually, a Dimension may contain multiple Hierarchies, such as Academic Unit-Department-Program and Academic Unit-Department. However, the SAS/MDDB is implemented so that each Hierarchy is in its own Dimension. Generally, access is through interfaces (such as LevelInterface or DimensionInterface from com.sas.mdtable) rather than through objects of the actual classes.

The "table" is defined by axes represented by the Axis class. Each Axis is defined by the Dimension(s) assigned to it. The Levels displayed are determined by the Hierarchy of the Dimension and any changes (such as drilling down or expanding) the user has made.

EXTRACTING MEMBER VALUES

To allow users to make choices that are used to pre-filter the data, the actual values must be displayed. These values are the Members of the Level.

To get a list of all of the Levels in an MDDB, use the getAllLevels() method of the MDDB model:

```
com.sas.collection.Dictionary mddb_levels = mddbModel.getAllLevels();
```

After this method call, mddb_levels will hold all of the individual levels of the MDDB, without regard to any hierarchies. The Dictionary object is keyed by the variable name in the MDDB:

```
LevelInterface entry_term_lvl = (LevelInterface)mddb_levels.get("entry_term");
```

This gives access to the Level object representing entry terms. The individual member values may be accessed as

```
MemberInterface[] entry_terms =
    entry_term_lvl.getMembers(0,entry_term_lvl.getMemberCount());
```

This array has all of the values of the underlying variable (in this case, entry_term) plus a placeholder for a total across all of the values (if totals are turned on). Note the use of interfaces to Level and Member instead of the implementing classes.

SUBSETTING THE DATA

Once the values are retrieved, they are used to populate HTML selection lists in a JavaServer Page:

```
<select name="select_term">
<% for(int i=0; i < entry_term_lvl.getMemberCount() - 1; i++)
{
%>
<option value="<%=entry_terms[i].getLabel()%>">
                <%=entry_terms[i].getLabel()%></option>
<%
}
%>
```

Similar code is used to retrieve and display the values for other variables from which the users can select.

Once the user makes a selection and submits their choices, control is passed to the MRV with the user selections as parameters on the request. The selections are retrieved from the JSP request object and then applied to the MDDB:

```
com.sas.collection.Dictionary mddb_levels = mddbModel.getAllLevels();
SubsetInterface entry_term = (SubsetInterface)mddb_levels.get("entry_term");
String[] term = {(String)request.getParameter("select_term")};
entry_term.setSubset(term);
```

Again, similar code is used to subset the data for other user choices. The result is a report like that in **Figure 2**.

Data source: Freshman Retention Data

Term Flag: Fall
Entry Department: Biology
Entry Term: Fall 2002

Term	Fall 2002	Fall 2003	Fall 2004	Fall 2005
	Enrolled Count	Enrolled Count	Enrolled Count	Enrolled Count
Term Department	Total Number of Nonmissing Values	Total Number of Nonmissing Values	Total Number of Nonmissing Values	Total Number of Nonmissing Values
Anthropo/Forensic Sci/Archaeo	.	.	.	7
Art	.	2	1	2
Biology	227	137	86	78
Business	.	9	11	5
Chemistry & Biochemistry	.	2	3	2
Communication Sci & Disorders	.	.	1	1
Communication Studies	.	1	3	3
Computer Science	.	2	.	.
Curriculum & Instruction	.	.	2	3
Economics	.	.	1	2
Education	.	5	.	.
Engineering	.	1	1	0

Figure 2: A Pre-Filtered MDDB Report

SUBSETTING HIERARCHIES

The approach outlined above works for simple `Levels`, but not for hierarchies. Subsetting on a hierarchy is equivalent to drilling down the hierarchy, and requires a different approach. The steps are

- Set the `Hierarchy` by name on the rows or columns of the report
- Set the drill subset on the first `Level` in the `Hierarchy` to the value selected by the user
- Get the `Axis` and position of the `Hierarchy` on the `Axis`
- Apply the drilldown to the Member representing the drill subset

To drill further down the `Hierarchy`, steps 2 – 4 are repeated with the second and subsequent levels in the `Hierarchy`.

To drill into the "Business" `Member` of the Unit-Program `Hierarchy`:

```
// Set the hierarchy
String row[] ={"Unit-Program"};
mddbModel.setRowAxis(row);

// Set drill subset on first level of hierarchy
level1 = (Level)levels.get("unit");
String drill_subsets[] = { "Business" };
level1.setDrillSubset(drill_subsets1);
 level1.drillSubsetNeedsRefresh();

// get the parent note of the hierarchy - this should be an Axis
ParentNode p = (ParentNode) level1.getRootParent();

//  position of hierarchy on axis
int hierPosn = level1.getAxisOffset();

// coordinate of drill value on axis
int valueCoord = ((Axis)p).getAxisCoordinate(drill_subsets1);
```

```
// get member to drill down on
Member m = (Member)((Axis)p).getLabels(valueCoord)[hierPosn];

// perform the drilldown
MDCommand command = new MDCommand(MDCommand.DRILLDOWN_COMMAND, m);
command.performAction();
```

CONCLUSION

By providing online access to interactive reports driven by MDDBs, IRT was able to place data directly in the hands of users and eliminate some ad hoc report requests. Additionally, the development of a way to pre-filter the data enabled users to quickly find the data they were looking for with minimal confusion.

REFERENCES

JavaDoc documentation for referenced packages and classes:

`com.sas.sasserver.mdtable`
http://support.sas.com/rnd/appdev/V2/webAF/api/com/sas/sasserver/mdtable/package-summary.html

`com.sas.servlet.beans.html`
http://support.sas.com/rnd/appdev/V2/webAF/api/com/sas/servlet/beans/html/package-summary.html

`com.sas.servlet.beans.mddbtable.commands`
http://support.sas.com/rnd/appdev/V2/webAF/api/com/sas/servlet/beans/mddbtable/commands/package-summary.html

`com.sas.servlet.beans.mddbtable.html.MDSelectorMenuItem`
http://support.sas.com/rnd/appdev/V2/webAF/api/com/sas/servlet/beans/mddbtable/html/MDSelectorMenuItem.html

`com.sas.servlet.beans.mddbtable.html.MDTable`
http://support.sas.com/rnd/appdev/V2/webAF/api/com/sas/servlet/beans/mddbtable/html/MDTable.html

ACKNOWLEDGMENTS

SAS Technical Support answered several questions about the use of the MDDB classes. They were particularly helpful in determining how to drill down a hierarchy programmatically.

RECOMMENDED READING

SAS webAF™ Component API JavaDocs: http://support.sas.com/rnd/appdev/V2/webAF/api/index.html

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Phil Rhodes
Baylor University
One Bear Place #97032
Waco, TX 76798
Work Phone: 254-710-8860
Fax: 254-710-2062
E-mail: Phillip_Rhodes@baylor.edu