SAP BW Reporting Framework: Virtual Provider Class based approach with BRF+ Integration

Applies to:
SAP BI  SAP BI 7.x. For more information, visit the Business Intelligence homepage

Summary
In this whitepaper I am going to explain how Virtual Providers can be build based on ABAP Classes. How BRF+ can be plugged in. And a framework that can be used to build real time analytics query.

Author: Debashis Ghosh
Company: SAP Labs India
Created on: 26 December 2012

Author Bio
Debashis Ghosh having 5 years of experience in IT, Associated with SAP Labs for last 3 years. During this tenure I worked with technical topics like BOL/GeNIL,BOPF,BI,CRM. I am working in a Product Development Team.
Table of Contents

SAP BW Virtual Provider Implementation Class Based Approach with BRF+ Integration...........................................3

Advantage of Using Class based Approach:..................................................................................3
- Extensibility ..................................................................................................................................................3
- BRF+ Integration for Value determination .................................................................................................4

Approach to build Reports with Framework (Concept).................................................................................5
- Example:- Plan Vs Actual Reporting Framework ......................................................................................5

Technical Steps to Follow for building Virtual Provider Class Based..........................................................7
- Output..........................................................................................................................................................14

Few technical details about BRF+ Implementation .........................................................................................14
- Extensibility Scenarios ..................................................................................................................................15

Special Note ....................................................................................................................................................15

Related Content .............................................................................................................................................16

Application of SAP BI InfoProviders ...........................................................................................................16

Copyright.........................................................................................................................................................17
SAP BW Virtual Provider Implementation Class Based Approach with BRF+ Integration.

For real time Analytical Reporting (like Plan Vs Actual report, Work In Progress report, Progress report) we can build Queries that is based on Multiprovider which in turn has Virtual Providers(Class based) underneath.

Say for example In Project System (PS) Scenario where there is a need to build a Query that reports about Plan Vs Actual cost. Where Planing data comes from Real time Info Cube and Actuals will come from Virtual Provider (Class based). Now multiprovider will combine these two data and Query can be built on top of that. For this topic Real time Info cube is not in the scope of our discussion.

Now to get the Actuals we can use Virtual Providers which can be built using ABAP Classes. Where we can use the full feature of ABAP. To get the Actual data we can write direct SQL queries or the Extractors (CO extractors for example) or we can call BRF+ functions to get the actual data from various DB tables, decision tables (in case of BRF+) in the class method. Once data is ready we can map the values with the corresponding Characteristics and Key Figures of the Virtual Provider. Hence we can take full advantage of ABAP OOPs also by this class based approach.

Advantage of Using Class based Approach:-

Extensibility

Objects Shipped by SAP:

1) Multiprovider.
2) Virtual Providers for various ERP Objects.
3) BRF+ Function with sample rules for value determination.

Customers are free to add any characteristics and Key figures to the SAP shipped Virtual Provider.

In order to fill the fields added by the customer, they have to inherit the SAP shipped Virtual Provider class and provide the logic to fill those fields and change the Virtual Provider class to their implemented class.

SAP will ship Virtual Provider Class which in turn implements IF_RSRDV_VPROV_LOC_NOSID_QUERY Interface.

Customers who inherit the standard Virtual Provider Class can

i) Call the parent class method using SUPER and add their own logic.
ii) Completely replace the parent class method implementation and write their own logic.
See the image below for Extensibility.

**BRF+ Integration for Value determination**

Another advantage is we can integrate BRF+ functionality in Virtual Provider. We can call BFR+ functions for value determination for characteristics and key figures.

Business Rule Framework plus (BRF+plus) provides a comprehensive application programming interface (API) and user interface (UI) for defining and processing business rules. It allows you to model rules in an intuitive way and to reuse these rules in different applications.

BRF+ Functions will be used. Value determination will help the customer to create rules that determines values for the fields.

Interface for the BRF+ Function is very generic that can be extended by the customer.
One Example of a rule could be as shown below.
A decision table expression could be used to build a rule as shown below.

<table>
<thead>
<tr>
<th>Cost Element</th>
<th>Activity Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>A1</td>
<td>RESOURCE</td>
<td>R1</td>
</tr>
<tr>
<td>C2</td>
<td>A2</td>
<td>RESOURCE</td>
<td>R2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost Element</th>
<th>Activity Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>A1</td>
<td>RESOURCE_TYPE</td>
<td>RT_1</td>
</tr>
<tr>
<td>C2</td>
<td>A2</td>
<td>RESOURCE_TYPE</td>
<td>RT_2</td>
</tr>
</tbody>
</table>

The output of the BRF function would be a name/value table where name will hold the field name and value will hold the field value.

The Virtual Provider class would read the name/value table and fill the respective fields with the value determined. (Technical details about implementing BRF+ is shared later in this topic)

**Approach to build Reports with Framework (Concept)**

**Example:- Plan Vs Actual Reporting Framework.**

**Steps followed for the Implementation of Plan Vs Actual Reporting.**

Probable approach for Plan Vs Actual Reporting using Virtual Provider and Real Time Info Cube

Step 1:-

Build Multiprovider by combining any Number of Virtual Provider with Real time Info Cube.
Step 2:
Build the Query and Reporting Layer on top of that, consume the Query from any UI.
Technical Steps to Follow for building Virtual Provider Class Based Approach for Virtual Provider we are taking the example of SFLIGHT.

Go to RSA1 txn and create the Virtual Provider with the Info Objects CARRIER ID, CONN ID, FLIGHT DATE, CURRENCY, PRICE.
Class Name for ex. Create CL_AVR_VPROV_C5169657_SFLIGHT and implement the interface IF_RSDRV_VPROV_LOC_NOSID_QUERY.

Redefine the three methods 1) Define
2) Read
3) Getfield (Blank implementation).
### Class Builder: Display Class CL_AVK_VPROV_CS160657_SFLIGHT

<table>
<thead>
<tr>
<th>Interface</th>
<th>Implemented / Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_AVK_VPROV_CS160657_SFLIGHT</td>
<td>Implemented / Active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abstract Path</th>
<th>Model Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_RSDRV_VPROV_LOC_NOSID_QUERY</td>
<td>Interface for Virtual InfoProvider Without SID In...</td>
</tr>
</tbody>
</table>

### Class Builder: Display Interface IF_RSDRV_VPROV_LOC_NOSID_QUERY

<table>
<thead>
<tr>
<th>Interface</th>
<th>Implemented / Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF_RSDRV_VPROV_LOC_NOSID_QUERY</td>
<td>Implemented / Active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_FIELDS</td>
<td>Static Method</td>
<td>Definition of Additionally Required Fields in t_data</td>
</tr>
<tr>
<td>DEFINE</td>
<td>Instance Method</td>
<td>Define the parameters</td>
</tr>
<tr>
<td>READ</td>
<td>Instance Method</td>
<td>Reads the Data</td>
</tr>
</tbody>
</table>

### Class Builder: Display Interface IF_RSDRV_VPROV_LOC_NOSID_QUERY

<table>
<thead>
<tr>
<th>Method parameters</th>
<th>Parameters</th>
<th>Type</th>
<th>P.O., Type</th>
<th>Associated Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_FIELDS</td>
<td>I_INFOPROV</td>
<td>Importing</td>
<td></td>
<td>REINFOPROV</td>
<td>InfoProvider</td>
<td></td>
</tr>
<tr>
<td>I_INT_FPC</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Class Builder: Display Interface IF_RSDRV_VPROV_LOC_NOSID_QUERY

<table>
<thead>
<tr>
<th>Method parameters</th>
<th>Parameters</th>
<th>Type</th>
<th>P.O., Type</th>
<th>Associated Type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE</td>
<td>I_TPS</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_TPS_FPK</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_AUSSN</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_KMARKTAB</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_KBEVAT</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_BMAX</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_BMAXMTS</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_VER_AGGREGATION</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_TPS_RSC</td>
<td>Importing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Create DS_SELECTION_PARAMETER
protected section.

```plaintext
types:
BEGIN OF s_selection_params,
  infoprov TYPE rsinfoprov,
  th_sfc  TYPE rsdri_th_sfc,
  th_sfk  TYPE rsdri_th_sfk,
  t_range TYPE rsdri_t_range,
  tx_rangetab TYPE rsdri_tx_rangetab,
  keydate  TYPE rrsrdate,
  maxrows  TYPE i,
  use_aggregation TYPE rs_bool,
  th_sfc_add TYPE rsdri_th_sfc,
  sel_fems TYPE rrkg_sel_fems,
END OF s_selection_params.

data DS_SELECTION_PARAMS type S_SELECTION_PARAMS.
  * s_selection_params .
```

We will be using this to buffer the characteristics and key figures to use this in other methods as these are only available in the `define()` methods, in order to access it from the `read()` method we need to buffer those parameters.

**IF_RSDRV_VPROV_LOC_NOSID_QUERY-DEFINE**

P.S:- Explore the parameters of this method in debug mode, like

- I_INFOPROV will give you the info provider name.
- I_TH_SFC will give you the name of the available characteristics in the infoprovder.
- I_TH_SFK will give you the name of the available Key Figures in the infoprovder.
- I_T_RANGE will give you the name of the available name value pair parameter table that is passed From SAP Exit variables.
method IF_RSDRV_VPROV_LOC_NOSID_QUERY~DEFINE.

    ds_selection_params-infoprov = i_infoprov.
    ds_selection_params-th_sfc   = i_th_sfc.
    ds_selection_params-th_sfk   = i_th_sfk.
    ds_selection_params-t_range  = i_t_range.
    * ds_selection_params-tx_rangetab = i_tx_rangetab.
    * ds_selection_params-keydate  = i_keydate.
    * ds_selection_params-maxrows = i_maxrows.
    * ds_selection_params-use_aggregation = i_use_aggregation.
    * ds_selection_params-th_sfc_add = i_th_sfc_add.
    * ds_selection_params-sel_fems = i_sel_fems.

endmethod.

if_rsdrv_vprov_loc_nosid_query~read

This method is used to read data to be filled for the info objects from the DB tables or from the Extractors.

One important things to notice is the parameter e_end_of_data

After mapping the values to the info objects which are read from the DB
We need to explicitly set it to e_end_of_data = abap_true.

METHOD if_rsdrv_vprov_loc_nosid_query~read.

DATA:
  lt_sflight  TYPE STANDARD TABLE OF sflight,
  ls_sflight LIKE LINE OF lt_sflight.

DATA:
  lr_data     TYPE REF TO data.

FIELD-SYMBOLS:
    <fs_sfc> TYPE rsdri_s_sfc,
    <fs_sfk> TYPE rsdri_s_sfk,
    <fs_sflight> TYPE sflight,
    <fs_data>  TYPE any,
    <fs_field> TYPE any,
    <ft_data>  TYPE STANDARD TABLE.

SELECT * FROM sflight
INTO TABLE lt_sflight.

IF sy-subrc EQ 0.
   DELETE ADJACENT DUPLICATES FROM lt_sflight COMPARING ALL FIELDS.
   SORT lt_sflight ASCENDING BY carrid.
ENDIF.

CREATE DATA lr_data LIKE LINE OF e_t_data.
ASSIGN lr_data->* TO <fs_data>.

LOOP AT lt_sflight
   ASSIGNING <fs_sflight>.

   LOOP AT ds_selection_params-th_sfc
      ASSIGNING <fs_sfc>.

      CASE <fs_sfc>-chanm.
         WHEN '/CPD/CARRID57'.
            ASSIGN COMPONENT <fs_sfc>-chaalias OF STRUCTURE <fs_data> TO <fs_field>.
            <fs_field> = <fs_sflight>-carrid.

         WHEN '/CPD/CONNID57'.
            ASSIGN COMPONENT <fs_sfc>-chaalias OF STRUCTURE <fs_data> TO <fs_field>.
            <fs_field> = <fs_sflight>-connid.
         WHEN '/CPD/FLDATE57'.
            ASSIGN COMPONENT <fs_sfc>-chaalias OF STRUCTURE <fs_data> TO <fs_field>.
            <fs_field> = <fs_sflight>-fldate.
      WHEN 'OCURRENCY'.
         ASSIGN COMPONENT <fs_sfc>-chaalias OF STRUCTURE <fs_data> TO <fs_field>.
         <fs_field> = <fs_sflight>-currency.
      WHEN OTHERS.
      ENDCASE.
   ENDDO.

   LOOP AT ds_selection_params-th_sfk
      ASSIGNING <fs_sfk>.

      CASE <fs_sfk>-kyfnm.
         WHEN '/CPD/PRICE57'.
            ASSIGN COMPONENT <fs_sfk>-kyfalias OF STRUCTURE <fs_data> TO <fs_field>.
            <fs_field> = <fs_sflight>-price.
         WHEN OTHERS.
      ENDCASE.
   ENDDO.

   APPEND <fs_data> TO e_t_data.
ENDLOOP.

  e_end_of_data = abap_true.
ENDMETHOD.
Output

After implementing the class Right click on the Virtual Provider and select display data.

```
"/CPD/AVR_FLT", List output
```

<table>
<thead>
<tr>
<th>...ARRID57</th>
<th>...ONNID57</th>
<th>...LDATE57</th>
<th>0CHNGID</th>
<th>Record typ...</th>
<th>0REQUID</th>
<th>Currency</th>
<th>...PRICE57</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>820</td>
<td>20.12.2002</td>
<td></td>
<td></td>
<td></td>
<td>CAD</td>
<td>1.222,00</td>
</tr>
<tr>
<td>AF</td>
<td>820</td>
<td>23.12.2002</td>
<td></td>
<td></td>
<td>EUR</td>
<td>2.222,00</td>
<td></td>
</tr>
<tr>
<td>LH</td>
<td>400</td>
<td>28.02.1995</td>
<td></td>
<td></td>
<td>DEM</td>
<td>899,00</td>
<td></td>
</tr>
<tr>
<td>LH</td>
<td>454</td>
<td>17.11.1995</td>
<td></td>
<td></td>
<td>DEM</td>
<td>1.499,00</td>
<td></td>
</tr>
<tr>
<td>LH</td>
<td>455</td>
<td>06.06.1995</td>
<td></td>
<td></td>
<td>USD</td>
<td>1.090,00</td>
<td></td>
</tr>
<tr>
<td>LH</td>
<td>3577</td>
<td>28.04.1995</td>
<td></td>
<td></td>
<td>EUR</td>
<td>6.000,00</td>
<td></td>
</tr>
<tr>
<td>LH</td>
<td>9981</td>
<td>21.12.2002</td>
<td></td>
<td></td>
<td>EUR</td>
<td>222,00</td>
<td></td>
</tr>
<tr>
<td>SQ</td>
<td>26</td>
<td>28.02.1995</td>
<td></td>
<td></td>
<td>DEM</td>
<td>849,00</td>
<td></td>
</tr>
</tbody>
</table>

Few technical details about BRF+ implementation

Now the BRF+ integration comes into picture. It can be plugged in as explained earlier, I will go through the technical details. We can call BRF+ functions from within the Read() method of the Class and can map the values read from the BRF+ based on the business needs.

I am providing here with a basic algorithm.

1) From BRF editor, obtain the Function id.
Data: lo_function type ref to if_fdt_function.
lo_function ?= cl_fdt_factory=>if_fdt_factory->get_instance( )->get_function( lc_function_id ).

2) Get the function context.
Data lo_context type ref to if_fdt_context.
lo_context = lo_function->get_process_context( ).

3) Set the input parameters for the BRF+ Function.
   Fill the value of ls_struct and pass it to the method.
   lo_context->set_value( 
       iv_name = 'ZINPUT_STRUC'
       ia_value = ls_struct ).

4) Process the function by passing the context received from step 2.

Data lo_result TYPE REF TO if_fdt_result .
lo_function->process( EXPORTING io_context = lo_context
                        IMPORTING eo_result = lo_result ).

5) Obtain the result
lo_result->get_value( IMPORTING ea_value = lt_tab ).

**Extensibility Scenarios**

1) Customer can disable SAP shipped rules and create their own rules.

2) Customer can extend the Input structure and append their own fields.

3) Using BRF+ Framework, customer can call Function Module/Static class methods as part of the rule.

4) Customers are free to provide their own logic to derive values by Inheriting SAP shipped standard Virtual Provider Class.

5) They can also create BRF+ functions and call them from inherited class.

**Special Note**

Example provided here is not implemented fully. It’s the framework that we followed based on product requirement, and I have given the outline to build real-time reporting, however different approaches can be followed on case basis. However the details about implementing Virtual Provider using Class based approach is covered in depth.
Related Content
You must include at least 3 references to SDN documents or web pages.

Application of SAP BI InfoProviders

Multi providers: Technical Details

Business Rule Framework Plus