Integration of SAP NetWeaver BRM with SAP BusinessObjects Query as a Web Service

Applies to:
SAP BusinessObjects Query as a Web Service XI 3.1 SP2 and SAP NetWeaver BRM 7.2
For more information, visit the Business Objects homepage.

Summary
This document describes the procedure to model business rules using data model represented by Query as a Web Service. This is a convenient way to consume SAP BusinessObjects Query as a Web Service in NW Composition Environment (CE) and BRM. This paper is written in collaboration with innovation-center.sap.com (http://innovation-center.sap.com).

The referenced files are available at the following location – http://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/a07eec61-babf-2c10-418e-fcf453ff0937

Author: Abhishek Kumar
Company: SAP Labs India, Ltd
Created on: 11 November 2009

Author Bio
Abhishek Kumar is a Project Lead in NW BRM team. He has been involved in development of Business Rules Management products for the past 5 years.
# Table of Contents

Introduction .................................................................................................................. 3

Business Scenario ....................................................................................................... 4

Required Software Components .................................................................................. 5

Step-by-Step Procedure ............................................................................................... 5
  Query as a Web Service ............................................................................................... 5
    Query as a Web Service query Generation ............................................................... 5
    Query as a Web Service Service Provisioning .......................................................... 6

SAP NetWeaver BRM .................................................................................................... 7
  Generating Business Vocabulary ................................................................................. 7
  Ruleset Modeling ......................................................................................................... 10
  Ruleset Web Services Provisioning ............................................................................ 12
  Deploying Rules .......................................................................................................... 13

Unified Service .............................................................................................................. 13
  Web Service client for Rules Service .......................................................................... 14
  Web Service client for QaaWS Service ........................................................................ 15

Conclusion ..................................................................................................................... 18

References ..................................................................................................................... 18

Copyright ....................................................................................................................... 19
Introduction

In an enterprise organization, data, using which rules are evaluated, often resides in one or more databases. BRM does not per se handle data; it only processes data that is given to it. This would mean that rules are actually written on data retrieved using database queries. But retrieving and combining data from diverse data sources and writing rules could become quickly very complex and painful task. A scenario such as this one takes away the very advantage that a BRM system can offer – facilitating the business user, without any technical expertise, to create, test and maintain business rules...

This adaptation and data mapping process has to be hidden as much as possible from the final consumer and should be as easy to implement as possible.

How does one solve the problem posed by the above use case? An ideal solution would be to present database queries as business terms that can be used directly in rules? Essentially, one would require a loose coupling between database queries and business rules. What it means is, the adaptation and data mapping process has to be hidden as much as possible to the final consumer and should be as easy to implement as possible.

This paper presents an overview and the concepts behind Business Intelligence and Business Rules integration. Both, Business Intelligence and Business Rules, have the ability to interact in a SOA environment and make these implementation requirements possible today.

While at one level, the integration described in this paper is pretty straightforward, it also shows some interesting points of integration between Business Rules and Business Intelligence:

- SAP BusinessObjects Semantic Layer (universe) as the Single source of the truth: BRM uses the same view on the Enterprise data that the business intelligence platform presents to users. This is an important assurance that Business Rules decisions and user decisions are based on the same facts.
- Nice separation of concerns with the rules engine executing rules and the business intelligence platform:
  - Business rules developer can quickly integrate data without having to worry about how potentially complex analytics are performed
  - Business Rules are easier to change than rules embedded in the BI platform
- No need for Business Rules developer to learn a new language to gather data like SQL or MDX - SAP BusinessObjects Semantic Layer helps you to quickly get analytics on any type of data source in a highly secured manner.
**Business Scenario**

Take a retail store ‘ABC’. The manager of this store offers discount for different product categories based on their price, quantity sold, revenue and cost of sales. All the relevant information is stored in the company database. Business Rules will be used to actually calculate the discount based on the different criteria.

To solve this problem, one would require the following services:
1. A service to securely fetch data from the Business Intelligence platform
2. A service which maps data to business terms that can be used to model rules, and also processes the data returned after rules execution
3. A service to execute rules on the data provided by service 2.

**Service 2** will be responsible for multiple tasks and is also vulnerable to frequent changes with the database queries. Since **Service 1** fetches data from the database that service 2 works with, there is a very tight coupling between Service 1 and Service 2.

The complexity involved in creating and maintaining the above services can be reduced by SAP BOBJ Query as a Web Service and SAP BRM. This approach would result in a solution that integrates data and rules without too much of code.

The integration of Query as a Web Service and BRM helps user to get object data without having any glue code in their application. Basically, this integration provides externalization of data instantiation part from the application.

**Query as a Web Service** is a tool from SAP BusinessObjects that can create and publish a query to fetch data from the database. The query is published as a web service which can be consumed by any other application.

Each query is exposed through a WSDL definition that contains all necessary metadata required to build the object model for the integration.

BRM allows modeling of rules based on a wsdl definition. The Ruleset with the modeled rules can then be published as a web service that can also be consumed by other applications. The Query as a Web Service and BRM integration would result in a unified service, a combination of the services published by Query as a Web Service and BRM.
Required Software Components

- SAP BusinessObjects XI 3.1 SP2 Enterprise
- Tomcat or another supported Web Server
- JDK 1.5 or above
- .NET 2.0 framework
- SAP NW Composition Environment

Step-by-Step Procedure

Query as a Web Service

Query As a Web Service is a tool from SAP BusinessObjects which helps user create and publish a service based on the query created from universe. A universe is a business representation of corporate data that helps end users access data autonomously using common business terms. In other words the universe represents database queries in the form of Business Terms which are easily understandable to Business Analyst and used for report generation and rules modeling. The SAP BusinessObjects Universe Builder helps in creating the universe from diverse data bases. In most scenarios the universes are already available as the same are used for reporting across the company. The user has to just create queries for the terms which are needed for rules modeling. More details can be found about Query as a Web Service at http://help.sap.com/businessobject/product_guides/boexir31SP2/en/xi31_sp2_qaaws_en.pdf

Query as a Web Service query Generation

We will use following business terms to create the query for our solution.

- **Unit Price MSRP**: This is the manufacturers suggested retail price per SKU and color.
- **Quantity sold**: Quantity sold - number of SKU sold
- **Sales revenue**: Sales revenue $ - $ revenue of SKU sold
- **Sold at (unit price)**: This is the actual unit price per SKU obtained at sale time (i.e. Revenue/Quantity)
- **Margin**: Revenue - Cost of sales
- **SKU**: Stock Keeping Unit number (SKU). The lowest level of product description.
- **Discount**: This is a virtual field and does not exist in database. This is added as query to get the discount result from the rules.

The universe which is used to create queries for above business terms is **eFashion** and this is available as an example universe with SAP BusinessObject XI3.1 SP2 installation.
Following screen shows the generated query:

Query as a Web Service Service Provisioning

See image below:
See the attached stockService.wsdl for the QaaWS service definition.

In the above wsdl the data for our business terms is represented by Table. The table has multiple rows and each row has SKU_number, Unit_Price_MSRP, Sales_revenue, Quantity_sold, Margin, Sold_at__unit_price_, and Discount as its child entities. The following XSD snippet shows, the way data is structured.

```xml
<xs:complexType name="Row">
  <xs:sequence>
    <xs:element name="SKU_number" type="xs:double" nillable="true" />
    <xs:element name="Unit_Price_MSRP" type="xs:double" nillable="true" />
    <xs:element name="Sales_revenue" type="xs:double" nillable="true" />
    <xs:element name="Quantity_sold" type="xs:double" nillable="true" />
    <xs:element name="Margin" type="xs:double" nillable="true" />
    <xs:element name="Sold_at__unit_price_" type="xs:double" nillable="true" />
    <xs:element name="Discount" type="xs:double" nillable="true" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Table">
  <xs:sequence>
    <xs:element name="row" maxOccurs="unbounded" type="s0:Row" />
  </xs:sequence>
</xs:complexType>
```

**SAP NetWeaver BRM**

The wsdl exposed by Query as a Web Service is used to model the rules. The XSD types of wsdl are used to generate aliases (Business vocabulary) for rules modeling.

**Generating Business Vocabulary**

**Importing a wsdl**

1. Create a Rules Composer DC in NetWeaver Developer Studio
2. In the Project Explorer view, expand the Rules Composer DC node, and the src node.
3. In the context menu of the wsdl node, choose Import.
4. In the wizard that appears, expand the Web Services node and choose wsdl. Choose Next.
5. In the screen that appears, choose Browse and specify the folder in your workspace where the wsdl file has to be placed and choose Remote Location / File System. Choose Next.
6. In the screen that appears, choose Browse and choose the wsdl file (stockServices.wsdl) in your system. Choose Finish.

**Adding the XSD Elements to the Rules Composer DC**

A wsdl file can contain multiple XSD elements, you can add the required XSD elements to the rules composer DC.

1. In the Project Explorer view, expand the rules composer DC node, the Rules Modeling node and double-click the Project Resources node.
2. In the Project Resources editor, choose the Aliases tab.
3. In the Aliases editor that appears, choose the Add button and in the menu that appears, choose XSD Element.
4. In the Add XSD Element dialog box that appears, expand the namespace node and choose the root element.
5. Choose “Create all default Xpath aliases for the selected element” radio button.
6. Choose Finish and save the changes.
The following screen shows how to add an XSD element for rules modeling. The XSD elements listed in the wizard are derived from the types of *stockService wsdl*.
The following screenshot shows the generated aliases for 'runQueryAsAServiceResponse' XSD type. The default alias name can be changed to give more user-friendly English name. The names are changed to Margin, Discount = (double), Quantity_sold in the below screen.
Ruleset Modeling

Once the aliases are generated the next step is to create the ruleset. The ruleset created for this solution is very simple and contains only one rule and one decision table. The decision table is used to calculate the discount for each product SKU number.

Creating a Ruleset

End of the example.

1. In the Project Explorer view, expand the rules composer DC node and in the context menu of the Rules Modeling node, choose New Ruleset.
2. In the dialog box that appears, enter the name of the ruleset (say stockRuleset) in the field. Choose OK.

Creating a Definition

1. Open the 'stockRuleset' editor
2. Navigate to Definitions page in the editor
3. Click on '+' icon under Variable Definitions.
4. Select ‘double’ as type from the drop down
5. Give the name of definition (say Cost_per_SKU)

Creating a Decision Table

1. In the context menu of the ruleset node ( stockRuleset), choose New Decision Table.
2. In the Decision Table Creation Wizard that appears, enter a name in the Decision Table Name (say discountCalculationDT ) field and optionally enter a description in the Comments field. Choose Next.
3. On the Select the Conditions screen, press Ctrl and select the alias ‘Quantity_sold’ and definition ‘Cost_PER_SKU’ in the Available Conditions section and choose the Select Conditions button.

The ‘Quantity_sold’ alias and ‘Cost_PER_SKU’ definition appear in the Selected Conditions section. Choose Next.

4. On the Select the Actions screen, press Ctrl and select the alias, Discount = {double}, in the Available Actions section and choose the Select Actions button.

The ‘Discount_sold’ alias appears in the Selected Actions section. Choose Finish.

5. Save the changes.
The following diagram shows the decision table

```
Decision Table : discountCalculationDT
```

### Documentation and Properties

<table>
<thead>
<tr>
<th>Quantity_sold</th>
<th>Cost_Per_SKU</th>
<th>Discount ={double}</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 50</td>
<td>&gt;= 4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Between 4 and 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>&lt;= 1</td>
<td>0</td>
</tr>
<tr>
<td>&gt; 50 and &lt; 200</td>
<td>&gt;= 3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Between 3 and 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>&lt;= 1</td>
<td>0</td>
</tr>
<tr>
<td>&gt; 200</td>
<td>&gt;= 2</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>Between 2 and 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>&lt;= 1</td>
<td>0</td>
</tr>
</tbody>
</table>

---

### Creating an If-Then Rule

1. In the context menu of the Ruleset Node (stockRuleset), choose **New Rule**.
2. In the dialog box that appears, enter a name (say stockRule) for the rule in the field. Choose OK.
3. In the rule editor that appears, under the **If** section, choose **(Add a new Condition)**.

   The default rule condition: `Operation.isSuccessful() Equals true` appears.

4. To edit the default rule condition:
   - Choose the LHS value: `Operation.isSuccessful()` and in the drop down menu choose a the alias `ns1:runQueryAsServiceResponse/ns1:table/ns:row.getXmlElement`.
   - Choose the comparator: **Not Equals** and choose a comparator in the drop down menu.
   - Choose the RHS value after the comparator and in the drop down menu choose the alias **null**. To enter static values, choose each component in the default rule condition (`Operation.isSuccessful() Equals true`) and enter the value in the inline text box.

5. In the rule editor that appears, under the **Then** section choose **(add a new Action)** and in the drop down menu, expand an action type node and choose **Assign :: Cost_Per_SKU action**. Choose the RHS value and create an expression: `(Sales_revenue − margin) / Quantity_sold`.

   Add one more action: **Evaluate-DecisionTable : discountCalculationDT**
The below screenshot shows the rule

```csharp
rule : stockRule
Priority : 50000
Overrides :
Effectivity : Always

<Click to enter comments>

Preconditions :
+

If
Sales summary record Not Equals null
+

Then
Assign :: Cost_Per_SKU = (Sales_revenue - Margin) / Quantity_sold.
Evaluate-DecisionTable :: discountCalculation
+
```

### Ruleset Web Services Provisioning

Once we have the Rule and the Decision Table ready, we would need to to publish this ruleset as a Web Service. We need to register SAP Java Server in NWDS to publish as a web service. The SAP Java Server can be configured in NWDS through preference page.

### Configure SAP Java Server

Follow below steps to configure SAP Java Server

1. In Preferences page, select SAP AS Java
2. Click on the Add and give **Instance Hostname** and **Instance Number** in the dialog
3. Click OK

### Rules as Web Service

Follow below steps to generate Web Service for stockRuleset

1. In the **Project Explorer** view, expand the rules composer DC and the **Rules Modeling** nodes.
2. In the context menu of a ruleset (stockRuleset), choose **Web Service ➔ Create WSDL Artifact**.
3. In the dialog box that appears, in the **Ruleset Name: Service Attributes** page, accept default values or make changes. Choose **Next**.
4. In the **Ruleset Name: Service Signature** page, the input and output types appear and the checkboxes are selected by default. Select all the check boxes. Choose **Next**.
5. In the **Ruleset Name: WSDL Preview** page, you should see the contents of the WSDL artifact.
6. Choose **Finish**
Deploying Rules
You should have a running instance of SAP AS, and should have configured the SAP NetWeaver Developer Studio with this instance.

1. In the Project Explorer view, in the context menu of the rules composer DC node, choose Development Component→Build.
2. In the dialog box that appears, select the rules composer DC checkbox and choose OK.
3. In the context menu of the rules composer DC node, choose Development Component→Deploy.
4. In the Deploy DCs dialog box that appears, select the rules composer DC checkbox and choose OK.
5. Open the NWA and navigate to ‘Web Services Navigator’
6. Search for the service ‘stockRuleset’ and open the wsdl from the link
7. Copy the wsdl to the file system.

See the attached rules.wsdl for the ruleset service definition.

Unified Service
Now we have both the services published. The next step is to create a client which unifies both the services. Since both the services work on the same type generating the proxy client is very simple.

The unification of these two services into one entity implies:

- An internal data mapping (at design time and runtime) between the Query web service output and the Business Rule web service input.
- That the input definition of the unified web service must include the input definition of the Query web service as well as the input definition of the Rule Engine web service unresolved through the above data mapping process. Optionally additional input definitions may be required to pilot the data mapping process.
- That the outputs of the unified web service can be empty or composed of full or partial Query and/or Business Rule web services output definition.

We will use SAP Java server to create client for both the services and finally create a single client which gets data from Query as a Web Service service which is then used to invoke Rules service.
Web Service client for Rules Service

Follow below steps to create a Web Service client for Rules Service

1. Create a Dynamic Web Project, say RulesService
2. Create a folder ‘src’
3. Create a folder ‘wsdl’ in the folder ‘src’
4. Copy the file ‘rules.wsdl’ to the wsdl folder
5. Right click on ‘rules.wsdl’ file and select Web Services -> Generate Client as shown in below diagram
6. Click **Finish** in the wizard which appears. You may choose to change the package name and set other configuration by going through next wizard pages. The wizard looks like

7. Clicking **Finish** will create the java client for rules service. See the attached java code.

**Web Service client for QaaWS Service**

Follow below steps to create a Web Service client for Rules Service

1. Create a Dynamic Web Project say **QaaWSService**
2. Create a folder ‘src’
3. Create a folder ‘wsdl’ in the folder ‘src’
4. Copy the file ‘stockService.wsdl’ to the wsdl folder
5. Right click on ‘rules.wsdl’ file and select **Web Services -> Generate Client**
6. Click **Finish** in the wizard which appears.
7. Clicking **Finish** will create the java client for rules service. See the attached java code.

**Unified Service**

Follow below steps to create a Unified Web Service.

1. Create a Dynamic Web Project, say **UnifiedService** and define project dependencies on above two projects (RulesService and QaaWS service).
2. Create a Java class and add below methods.
3. Expose the method ‘getTableInforFromRules’ as a webservice.
The below code snippet shows how we can combine both the services.

```java
/**
 * This method fetches data for Table element from QaaWS service.
 */
public static Table getTableInfoFromQaaWS()
{
    try {
        URL wsdlUrl = new URL("http://inln50076293a.dhcp.blrl.sap.corp:8080/dswsbobje/qaawsservices/?WSDL&cuid=AVYcPaCLNsJPK7v_53vcv6U");
        QName qname = new QName("storeService","storeService");
        StoreService service = new StoreService(wsdlUrl,qname);
        QueryAsAServiceSoap queryAsAServiceSoap = service.getQueryAsAServiceSoap();
        RunQueryAsAService parameter = new RunQueryAsAService();
        parameter.setLogin("Administrator");
        parameter.setPassword("abhishek123");
        QaaWSHeader header = new QaaWSHeader();
        header.setSerializedSession("aaa");
        header.setSessionID("bbb");
        RunQueryAsAServiceResponse runQueryAsAServiceResponse = queryAsAServiceSoap.runQueryAsAService(parameter, header);
        return runQueryAsAServiceResponse.getTable();
    } catch (MalformedURLException e) {
        //do nothing
    }
    return null;
}

/**
 * The method combine the service call for QaaWS and Rules service and get the final
discount from the
 * rules. The value of Table element retrieved from QaaWS (in the method
 * "getTableInfoFromQaaWS") is
 * mapped to the Table element which is passed as input to rules service.
 */
public static Table getTableInfoFromRules()
{
    try {
        QName qname = new QName("http://www.sap.com", "stockRuleset");
        StockRuleset stockRuleset = new StockRuleset(wsdlUrl,qname);
        StockRulesetPortType stockRulesetPort = stockRuleset.getStockRulesetPort();
        RunQueryAsAServiceResponse runQueryAsAServiceResponse = new RunQueryAsAServiceResponse();
        //Fetch the information of Table to from QaaWS service
        Table table = getTableInfoFromQaaWS();
        response.setTable(table);
        //Pass the information of Table to rules service
        RulesTypesDemoSapComStockrulesStockRulesetStockRuleset parameter = new RulesTypesDemoSapComStockrulesStockRulesetStockRuleset();
        parameter.setRunQueryAsAServiceResponse(response);
        RulesTypesDemoSapComStockrulesStockRulesetStockRuleset invokeRules = stockRulesetPort.invokeRules(parameter);
        RunQueryAsAServiceResponse runQueryAsAServiceResponse = invokeRules.getRunQueryAsAServiceResponse();
```
Table tableResponse = runQueryAsAServiceResponse.getTable();
//Get the final Table value after rules execution
List<Row> rowsRules = tableResponse.getRow();
for(Row row : rowsRules){
    System.out.println(row.getSKUNumber());
    System.out.println(row.getDiscount());
}
} catch (MalformedURLException e) {
    e.printStackTrace();
} return null;

Please see the attached java code for unification of the Query as a Web Service and Rule services.

The below table shows the final discount obtained for some of the products after the execution of the service

<table>
<thead>
<tr>
<th>SKU Number</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>115121.0</td>
<td>1.5</td>
</tr>
<tr>
<td>116256.0</td>
<td>3.0</td>
</tr>
<tr>
<td>119427.0</td>
<td>3.0</td>
</tr>
<tr>
<td>120114.0</td>
<td>3.0</td>
</tr>
<tr>
<td>121764.0</td>
<td>3.0</td>
</tr>
<tr>
<td>122709.0</td>
<td>3.0</td>
</tr>
<tr>
<td>128390.0</td>
<td>2.0</td>
</tr>
<tr>
<td>128969.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Conclusion
Combining both Query as a Web Services and BRM as one entity hides the complexity of data mapping and its flow from business user. It also offers simplicity through semantic translation of the Query output data model to the Rule Engine input definition and simple data mapping allowing the final integration to increase the level of information.

References
For more information on SAP BusinessObjects Query as a Web Service
See also Web Intelligence chapter “Sharing Web Intelligence content with other Web applications”

For more information on NW CE BRM
Integration of SAP NetWeaver BRM with SAP BusinessObjects Query as a Web Service

Copyright

© Copyright 2009 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System z, System z10, System z9, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, z/VM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

BusinessObjects and the BusinessObjects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other BusinessObjects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of BusinessObjects S.A. in the United States and in other countries. BusinessObjects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies (“SAP Group”) for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.