IBM Operational Decision Manager

Version 8.6.0

Patterns for combining Analytics with Operational Decision Management in Smarter Processes

This edition applies to version 8, release 6, modification 0 of Operational Decision Manager and to all subsequent releases and modifications until otherwise indicated in new editions.

Abstract

With the advent of mobile, cloud and big data processing, businesses need to respond quickly to emerging risks and opportunities as part of a smarter decision-making process.
This article looks at patterns that allow business insights and situational awareness provided by big data and analytics to be combined with Operational Decision Management to produce actionable responses in business solutions.
A retail scenario illustrates how IBM Operational Decision Manager can be integrated with IBM SPSS predictive analytics, IBM Integration Bus and IBM Business Process Manager, to perform discounted pricing of books based on emerging customer characteristics.
Recommended practices are described so that solution architects and integrators understand how these products can be used together. While the article describes many of the key installation, configuration and development tasks for a solution, administrators, analysts and rule developers should refer to the appropriate product documentation. This paper describes how Operational and Analytical decision management products can be integrated together to realize smarter processes and solutions.

Authors: *Duncan Clark, Katherine Tsui, Gavin Willingham, Lucinda Croft, Peter Seddon*

Technology:
Operational Decision Manager 8.6
IBM SPSS Collaboration and Deployment Service 6.0.0.1
IBM SPSS Modeler Server 16
IBM Analytical Decision Management 8.0
IBM Integration Bus 9.0.0.2
Business Process Manager 8.5.5

Level: *Beginner/Intermediate*

Contents

Decision management integration patterns overview

Overview

Business Rule Management systems have been evolving over many years to provide a means of automating frequently occurring decisions that are required to make day-to-day operations run effectively. These decisions ensure that customers are treated consistently, that the right price is offered or that the most effective offer is made.

Business decisions are often based around policies on how an organization should conduct its business to better meet business goals or to conform to regulations.
IBM Operational Decision Manager allows organizations to capture these decisions in order to automate them as decision services. These services can then be used to improve the straight through processing and increase effectiveness and operations efficiency consistently across the organization.

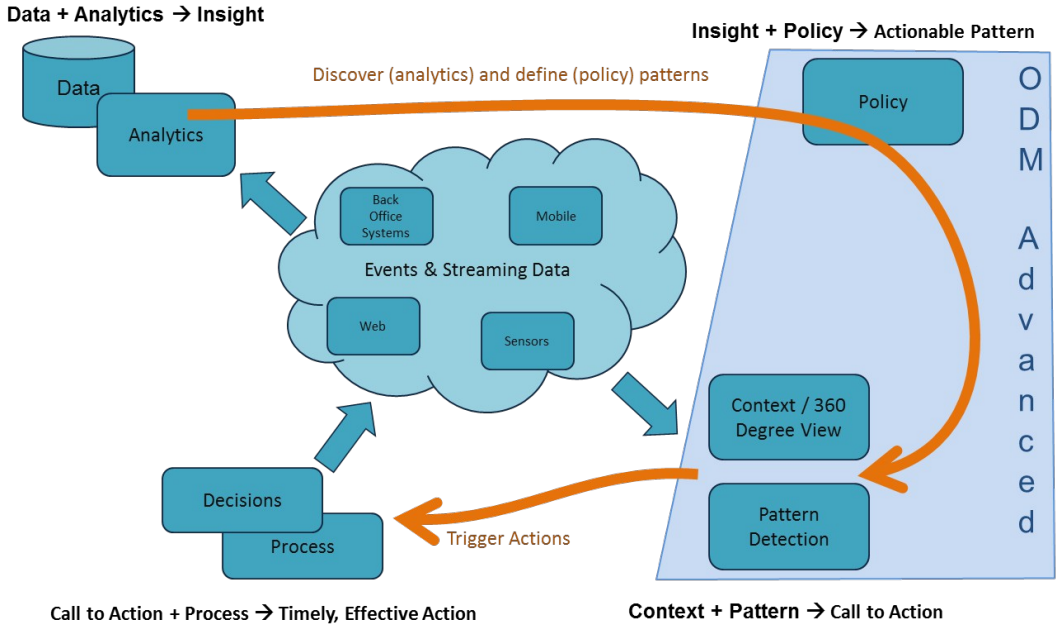
Traditional approaches focused on recording and manipulating records using synchronous decision services to define the actionable response as part of a well-defined process. With the advent of mobile, cloud and big data processing, businesses need to respond to emerging risks and opportunities at the earliest actionable moment within a smarter decision-making process.

Risk or opportunity can be considered by asynchronously accumulating information as a basis for identifying the situations and context in which the action needs to be taken. While the reasons for actions in traditional approaches are obvious and may be defined explicitly in the rules, this new approach is much more subtle, evolves over time and requires the use of insights provided from analytics and big data in order to make an effective response.

Article scope

This article is one of a series of articles that look at patterns that allow business insights and situational awareness provided by big data and analytics to be combined with Operational Decision Management to produce actionable responses in business solutions.
Figure 1 shows the overall concepts that will be discussed in these articles.

Figure 1 High level operational view



In figure 1 you can see how the insight derived from data and analytics allows the behaviour of solutions to be understood and policies to be discovered and defined. Operational Decision Management rules and pattern matching techniques can then be used with the information coming out of the analytics to establish the 360 degree view of current evolving situations and thus trigger actionable responses at the earliest opportunity. Processes and traditional decision services can then be used with this improved context information to optimize the response to the situation.

Solution context

This section describes the key product components and integration points that can be combined to produce these actionable insight solutions. The patterns that will be covered by this series of articles are:

- Patterns for integrating operational decisions into Smarter Processes (pending publication) – shows the basic patterns for integrating operational decisions into solutions
- Patterns for operational and analytical decision management in Smarter Processes (this article) - shows how operational decisions and predictive analytics can be leveraged in solution
- Patterns for integrating operational decisions into streams and Big Data solutions (future) - shows how operational decisions can be leveraged as part of an analytical insight solution
- Patterns for actionable insight (future) shows how IBM Operational Decision Manager Advanced can be integrated into these solutions

Figure 2 shows the overall solution context and integration points between the products and components in these patterns.

Figure 2 Solution context diagram

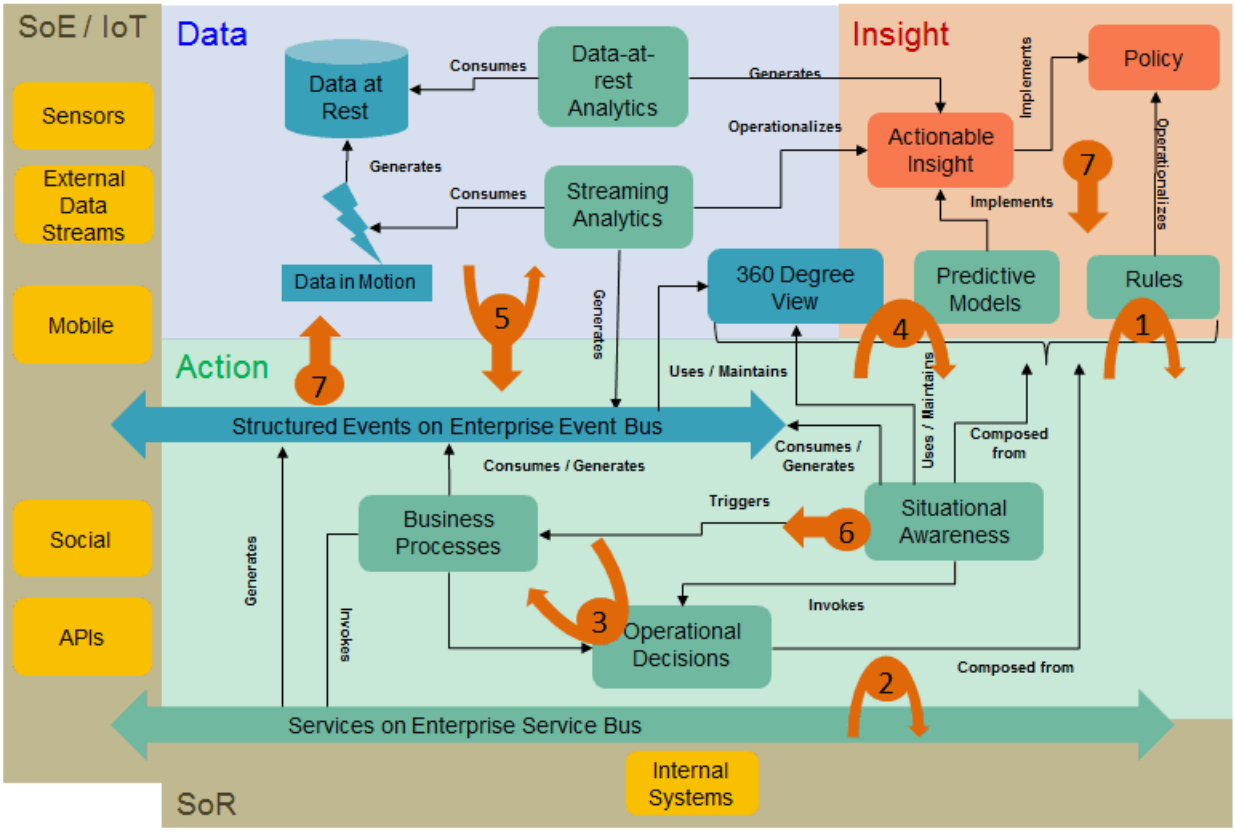


Figure 2 shows five main areas of actionable insight solutions.

- Systems of Record (SoR) - provide the internal systems (databases, transaction processing systems) that perform the main business of the organization. These systems are exposed through services, messaging infrastructures or Enterprise Service Bus's to:
- Systems of Engagement (SoE/IoT) – provide the multichannel access for the business solutions to partners and customers. This area is growing very fast supported by Mobile, Cloud and the Internet of Things (IoT) and is also generating large amounts of information that can be leveraged by:
- Data – provides the means to assimilate and gain insight from the large amounts of data held in Big Data repositories (Hadoop, Big Insights – data at rest) or streams of information coming from sensors or social applications (data in motion). Analysis of this data provides:
- Insight – captured as predictive models or policies (rules) specified according to what business analysts have learned from their solutions. This Insight is then leveraged to realize:
- Action – where the emerging situation can be used to trigger processes and tasks at the appropriate time, making decisions based on up-to-date, complete and precise information.

Figure 2 also highlights the key integration points between IBM Operational Decision Manager and other products / components used in an actionable insight solution.

The following integration points are covered in this article with an emphasis on integration with analytics (point 4):

1. Operational Decision Manager decision services where the policies defining what people know can be expressed as rules and automated.
2. Decision service integration using an Enterprise Service Bus where the decisions can be turned into actions in the context of the business solution.
3. Making insightful decisions as part of smarter processes where the flow of process activities and tasks can be automated according to the decisions.
4. Leveraging 360 degree insight and predictive analytics in operational decision services allowing a more accurate decision at the time the decision is made.

The following integration points will be described in future articles:
5. Applying rules based decisions in Big Data and streams based processing
6. Situational Awareness and Action using Operational Decision Manager Advanced
7. Decision service monitoring, simulation and improvement

Each numbered integration point is now described in more detail in the following sections.

1. Operational Decision Manager decision services

The starting point for this series of articles is the synchronous decision service capabilities provided by IBM Operational Decision Manager. These can be integrated into applications or processes using the patterns described in this series of articles. The behavior of the decision services can be controlled by the business and evolved using rules and decision tables to meet the evolving policies. It is not the intention to describe the detailed capabilities of IBM Operational Decision Manager but it is important to understand the underlying principles shared across the integration patterns.
Readers should refer to the IBM Operational Decision Manager Knowledge Center for further information about IBM Operational Decision Manager.

2. Decision service integration using an Enterprise Service Bus

IBM Integration Bus provides a flexible environment for implementing both Event and Enterprise Service Buses. This integration point shows how IBM Integration Bus can virtualize decision services or event based interactions as well as leveraging other sources of information used in the decision making process.

Customers also require well defined APIs for exposing their decision services to the broader Systems of Engagement (SoE) or Internet of Things (IoT). These environments now require REST / JSON services for use from mobile devices or from applications on the cloud. The information needed to make the decisions is drawn from a wide variety of sources and formats

across the cloud based ecosystem requiring flexible yet robust decision service API management based on these underlying virtualized decision services.

Readers should refer to the IBM Integration Bus Knowledge Center for further information about IBM Integration Bus.

3. Making insightful decisions as part of smarter processes

Insight from "what your data knows" and "what your organization knows" drives the "action" in actionable insights. These policies and insight are used in decision making through the decision services or other pattern matching techniques. The decisions made influence the actions taken by the organization as part of their day-to-day activities and processes.

Insight can be used in two key areas in smarter processes:

- Deciding when to act – situations that start, progress or cause an exception path to be adopted in a process.
- Making an insightful decision to decide on the next activities to undertake within the process.

This combination of situational awareness and insightful decisions is what allows the actions to be taken in the business moment.

Readers should refer to the IBM Business Process Manager Knowledge Center for further information about IBM Business Process Manager.

4. Leveraging 360 degree insight and predictive analytics in decision services

Big Data and analytics are now able to provide deep insights and 360 degree information about entities (erg. customers, products) that are important to the business. Predictive models based on historical analytics then allow predictions to be made of future customer behavior allowing decisions to be made with the advantage of "hindsight" and thus the business outcome optimized.

Big Data and analytics not only provides insight into the behaviors of customers and the potential market but also allows the overall performance and trends of the business to be monitored and visualized through dashboards and reports. Using this business status information allows situations to be detected at an early stage and corrective action applied before KPIs degrade.

Consumers of decision services want to consider the latest situational awareness and predictive analytics when making decisions. This information is not directly available to those consumers and has to be drawn either from the cashed situational state or directly from the analytics. Virtualization techniques such as IBM Integration Bus, API
Management or even the integration services in Business Process Manager can be used to ensure that the decision leverages this evolving insight.

The IBM SPSS Collaboration and Deployment Services Information Center describes how to build and develop analytical models to produce this insight. The IBM Analytical Decision Management

Information Center describes how these models can be used to make analytical decisions based on these models. This article also describes how IBM Operational Decision Manager can be leveraged as part of an Analytical Decision Management application but does not describe the capabilities of Analytical Decision Management solutions.

5. Applying rules based decisions in Big Data and streams based processing

Social computing and the Internet of Things are leading to massive quantities of information being made available to organizations. To analyze and process this information, technologies such as streams processing and Hadoop are applying massively parallel processing close to the data. The use of decision services whose behavior can be configured by the business to classify and filter this information is becoming more and more important to identify emerging situations that require attention.

Readers should refer to the IBM Infosphere Streams Knowledge Center and IBM InfoSphere BigInsights Knowledge Center for further information.

6. Situational awareness and action

Situational awareness means knowing when to act – by bringing together analytic insight and rules to describe the situations – combinations of past events that happened, events that didn't happen, current state, and predictions that demand immediate attention.

Just in time awareness of risk and opportunity – the ability to detect any situation immediately upon receipt of the information that "concludes" the situation – is the bridge from insight to action, and triggers action customized to individual risks and opportunities. In the general case, action is process – a straight through orchestration, workflow or case management response to the situation.

7. Decision service monitoring, simulation and improvement

The key goal of decision services and complex rule based event processing is to allow the business to express and manage the required behavior of their solutions using rules. By monitoring and analyzing the behavior of the business in response to the decisions, organizations can understand how the rules and policies that are used in the decisions affect the business.

Once organizations have this insight on how their decision making affects the business, they can start to optimize their business by careful change management of those rules and policies. Simulation of decision making based on historic data records is often used to evaluate the effectiveness of new policies requiring close integration between data, decision management and KPIs and dashboards.

Analytical and operational decision management patterns overview

The goal of this article is to describe how operational decision management solutions can leverage analytics information to make better decisions. The article also describes how analytical decision management applications can leverage business rules from IBM Operational Decision Manager. The article is arranged as five patterns:

- Pattern 1 – Operational Decision Making – describes how operational decision services should be designed to be able to leverage insight within the rules.
- Pattern 2 – Analytical Decision Making – describes how operational decisions can be included in an Analytical Decision Management application.
- Pattern 3 – Predictive Scoring Services – provides an overview of how an SPSS analytics scoring model is developed and accessed.
- Pattern 4 – Enterprise Service Bus integration – describes how IBM Integration Bus can be used to aggregate the information from analytics and scoring to pass to the decision service as part of a solution message flow.
- Pattern 5 – Smarter Process integration – describes how IBM Business Process Manager can be used to aggregate the information from analytics and scoring to pass to the decision service as part of a business process.

The next section describes a retail business scenario that illustrates the use of these patterns.

Business scenario and use case

This section describes a simplified retail scenario that illustrates how IBM Operational Decision Manager can be integrated with IBM SPSS predictive analytics, IBM Integration Bus and IBM Business Process Manager, to perform discounted pricing of books based on emerging customer characteristics.

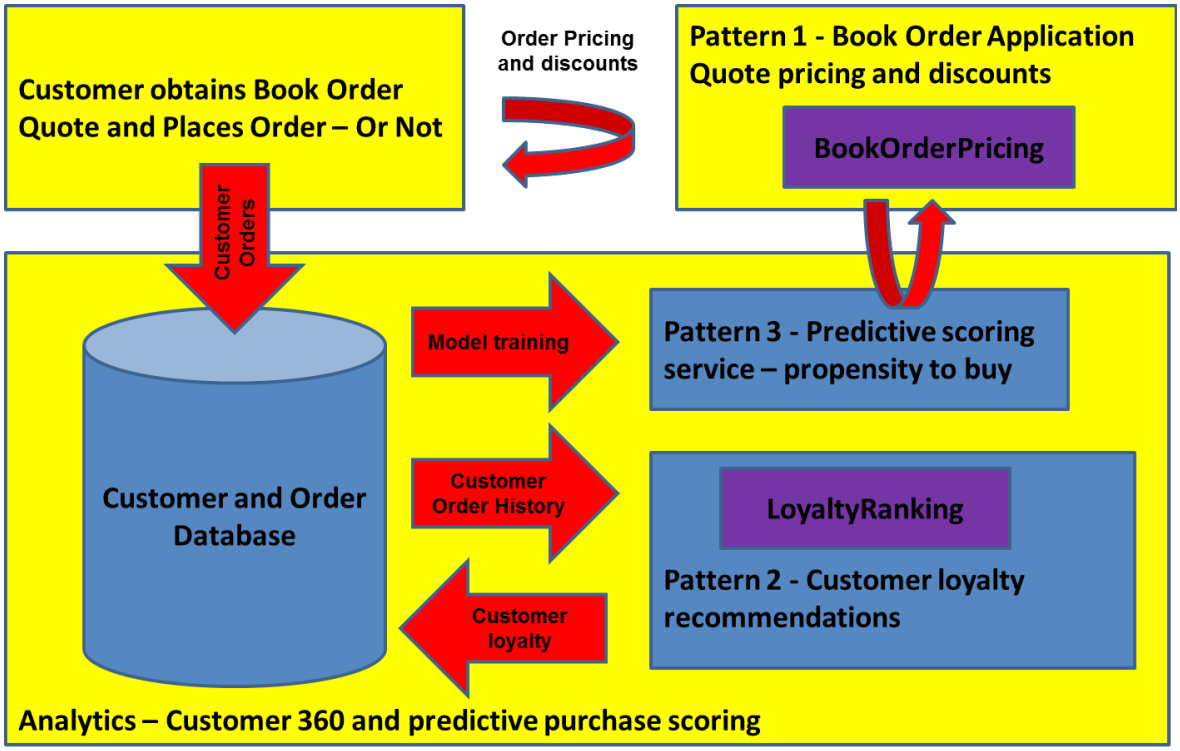
Scenario overview

The scenario is based around a book retail organization. The organization provides their customers with a number of channels to buy their products. Products can be discounted based on the quantity of products being bought, the loyalty of the customer and any marketing plays that are being exercised.
The scenario focusses on the interactions between Operational and Analytical Decision management within a solution that could be based either on an IBM Integration Bus message flow or as part of a Business Process Management book order process.

A secondary scenario (pattern 2) describes how Analytical Decision Management can leverage Operational Decision Manager when considering loyalty card recommendations. This article does not describe the use of predictive analytics in this scenario as this is expected to be well understood by Analytical Decision Management practitioners.

A high-level overview of the customer solution is shown in figure 3 below identifying the three main integration patterns to be described.

Figure 3 Scenario Overview



After browsing through a book catalog, the customer can create a book order specifying the products and quantities that they require and any promotion code they wish to use.

The Book Order Application then shows how the quote price for the order is then calculated used an operational decision service (pattern 1). This decision is based on the price of individual books but includes personalized discounts that are applicable to this customer based on the 360 degree information available for that customer. The decision is also based on a predictive model that determines the likelihood of the customer placing an order for this quote. This likelihood can be used to influence the price and discounts offered and thus improve the chance of the customer accepting the quote and placing and order.

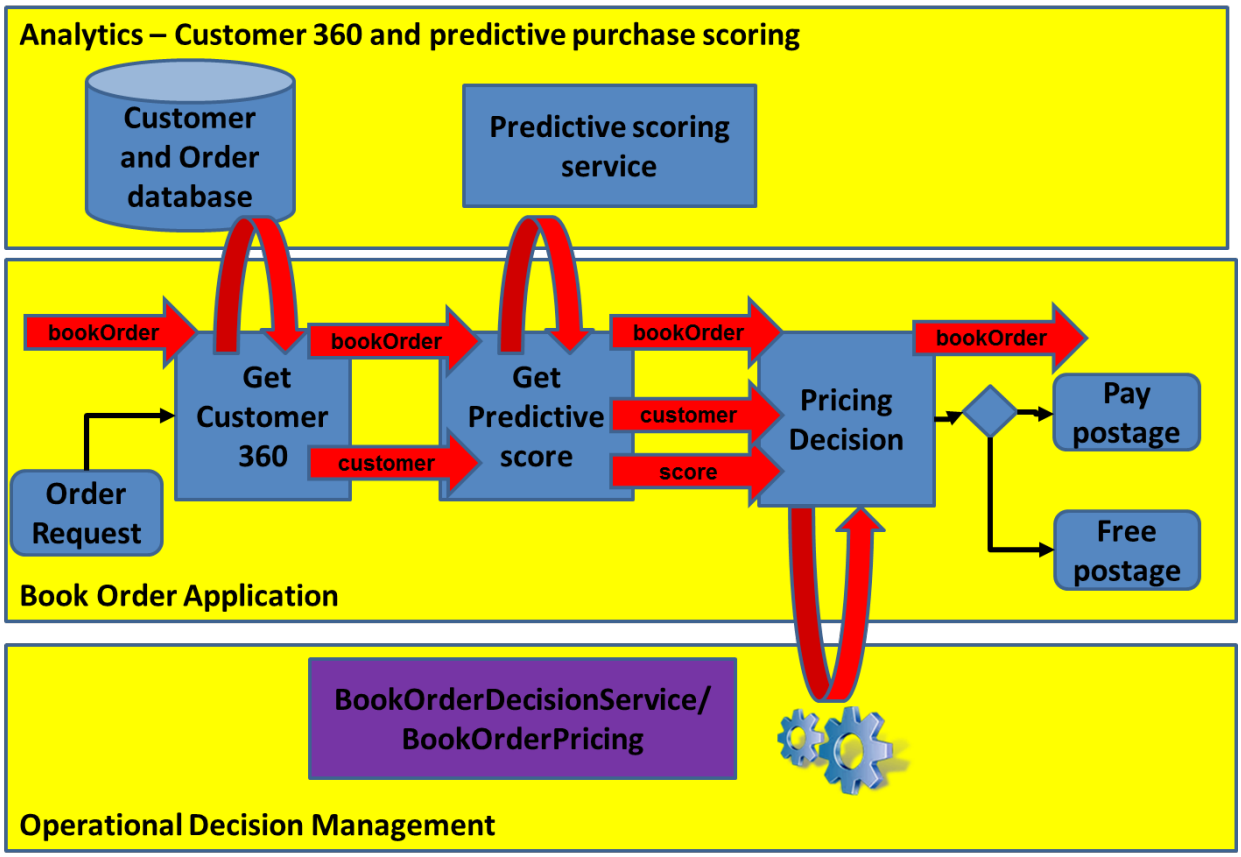
After the order quote is offered, the customer may or may not place an order. At this point the analytics scenarios start. Each order (or quote) is stored in a Customer and Order Database (BKORDER) together with the information about whether the order was actually placed or not.

A Customer Loyalty Recommendation application (Analytical Decision Management - pattern 2) then correlates orders from a particular customer, and determines their loyalty. As part of this analysis a Loyalty Ranking decision service is called to recommend the loyalty for a particular customer based on their order history. This information can then be used in the analytics applications for making loyalty card recommendations or other discount offers.

The analytics processing also includes a predictive model used to calculate the likelihood of a customer placing an order (Predictive scoring services - pattern 3). This has to be based on historical experience and can use the information in the Customer and Order Database (BKORDER) to train the model and thus provide a predictive scoring service that can indicate the likelihood of the customer placing an order based on customer characteristics.

The Book order application is illustrated in figure 4 and shows how the information is obtained in order to make the pricing decision.

Figure 4 Operational decision management patterns



The Book Order application shows how an Operational decision service (BookOrderDecisionService - pattern 1) can use the customer 360 degree information and predictive models to improve the pricing decisions. The Book Order application may be implemented as an IBM Integration Bus Message flow (pattern 4) or a Business Process manager flow (pattern 5).

On placing the order request, the customerID is used to retrieve the 360 degree information about the customer. This includes their age, gender and loyalty.

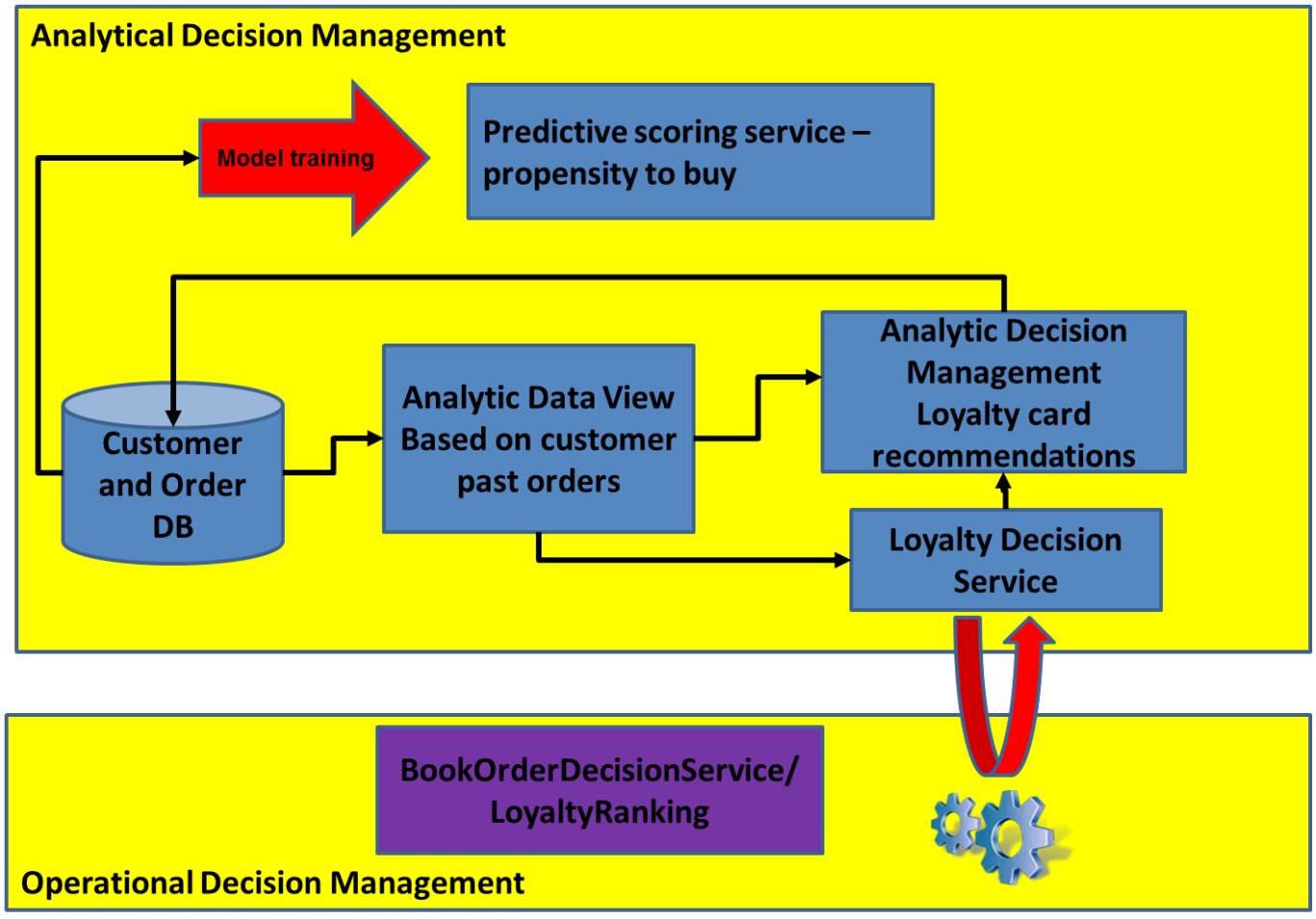
The application then has to retrieve a predictive score that indicates the likelihood of the customer accepting a quote. This might be based on characteristics of the customer and order. In this scenario the prediction is based simply on customer gender and loyalty as described in pattern 3.

This information is all passed to the BookOrderPricing decision which then calculates a price and appropriate discounts using rules. The likelihood of the customer placing the order is used to determine whether to offer free post and packing or not. Customers that are likely to reject a quote are offered free post and packing in addition to their other discounts.

Based on a value for post and packing, the application can then route the quote through a different channel for free or paying postage customers.

The customer can then decide whether to place the order and this information can then be used in the Analytical decision management patters described in figure 5.

Figure 5 Analytical decision management patterns



The Analytical decision making pattern (pattern 2) describes how the analytics can leverage rules and the Loyalty Ranking decision service operation to help determine which customers should be offered certain promotions. The scenario provides a DBCOWE promotion code that is offered only to Gold customers. This analytics pattern shows how an Analytic Data View can be produced that aggregates the order history for any given customer. This view can be passed to IBM Operational Decision Manager to make a recommendation as to the customer loyalty which can then be used in the Analytic Decision Management application to determine who should receive which offers and thus become eligible for the DBCOWE promotion.

Finally the history of orders placed against customer characteristics can be used to develop a predictive model (pattern 3). This indicates whether the customer is likely to place an order based on their gender and loyalty. In a real scenario the likelihood of purchase will be based on many factors which will not be considered in this simple scenario.

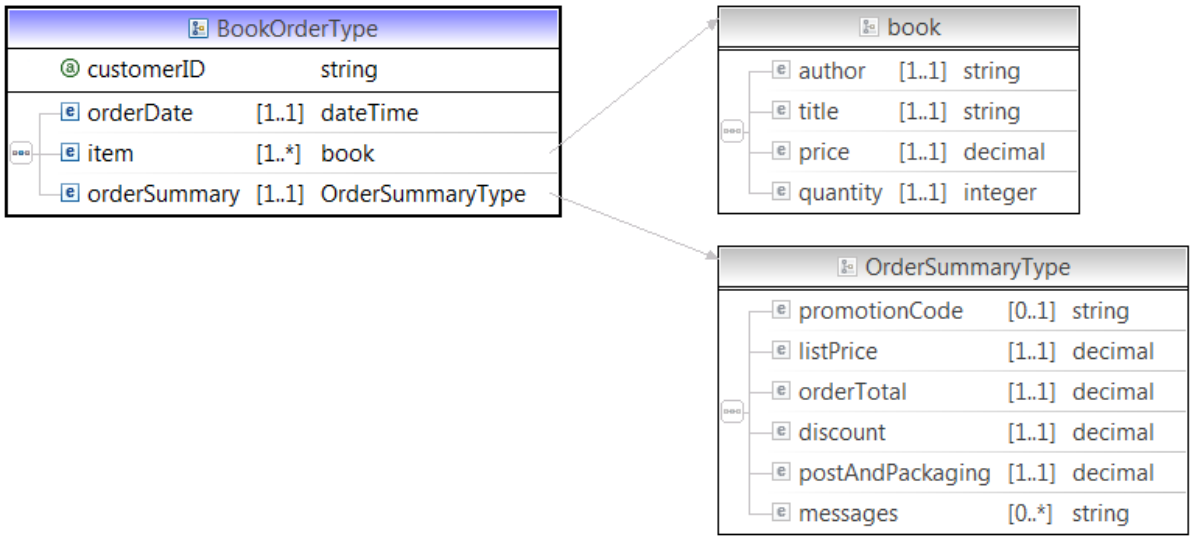
Once a predictive model has been built, this can be exposed as a scoring service which can be invoked as part of the operational decision making. As more customers make purchases, this model can be refined and the accuracy of the predictions improved

Information model

This section highlights the key objects used in the solution and the important fields in their representation. In a real solution, there will be a number of different models used by different parts of the solution. In this sample we have used two XML models.
One is based on the main operational interactions as described in this section. This concentrates on the information needed to provide the quote and includes additional fields needed to represent the customer 360 view and scoring service.
A second model has been derived from the SPSS data sources being used in the analytical modeling. This is described in the Analytical Decision Making pattern section.

Figure 6 describes the top level Operational Decision Making model.

Figure 6 Book Order information model



The BookOrderType is used to represent the basic order passed to the decision service. The customerID attribute indicates the identity of the customer which is used to retrieve the latest 360 degree information about that customer when making the pricing decision.

Each BookOrderType contains an orderDate time stamp and a list of books being ordered. Each book item contains a price and quantity (as well as name and author) that must be populated prior to the pricing decision.

The main details returned by the decision are provided in the orderSummary.
The order pricing may be based on additional fields in the orderSummary:

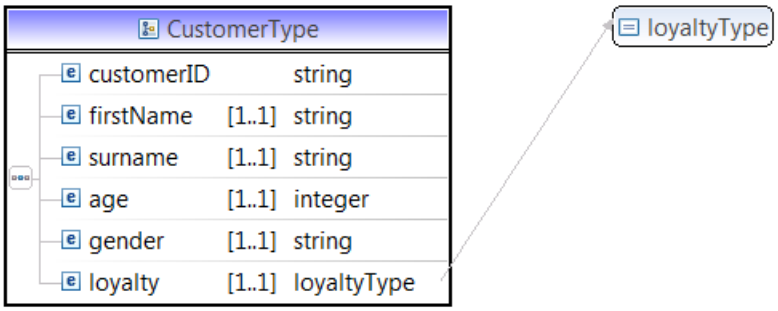
- promotionCode entry supplied by the calling application / customer indicating any special promotions that they wish to apply to the order. As promotion codes are only relevant to certain customers, this information may also be updated by the rules.

The rules in the decision then determine:

- promotionCode determines any promotionCode actually applied to the order-
- listPrice is calculated from the total prices of the books in the order
- orderTotal - the total price of the order after discounts but excluding post and packing
- discount – the total discount applied to the list price
- postAndPackaging - denoting the post and packing charges applied. This may cause routing decisions in the parent flow or process
- messages - denotes a list of applied discounts and offers included in the price

In order to make effective decisions, the rules also need information about the customer. This is provided as a separate class as shown below.

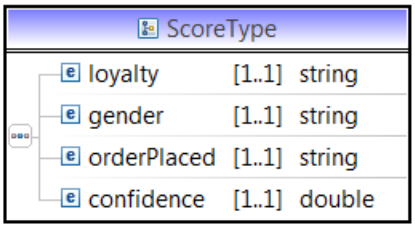
Figure 7 Customer information model



The customer information model defines the customers name and other information that may be used to make informed pricing decisions.
The age and gender would be derived from customer records while the loyalty is based on analytics and derived from the customer's spending. The loyalty is therefore a good historical indicator of the customer's propensity to buy and thus serve as a basis for the need to offer discounts in order to close a order.

In this scenario we will use the customer gender and loyalty to call a predictive model that indicates if the customer is likely to place an order based on this quote.

Figure 8 Predictive Score Information Model



The ScoreType represents the information used to obtain a prediction of whether this customer would place an order (orderPlaced = Y) based on their gender (M/F) and loyalty (NONE/BRONZE/SILVER/GOLD). This scoring information is passed to SPSS to obtain a score based on customer's historical records.

Scenario Patterns

This section describes provides an overview of the patterns that are described in more detail in the rest of this article.

Pattern 1 – Operational Decision Making

This interaction describes how IBM Operational Decision manager provides a BookOrderPricing decision service that provides the pricing logic. The pattern defines the book order information (including the analytical information) that should be supplied to make the decision, the rules that are applied and the book order summary information that is returned. This decision service can be accessed by any application using a variety of the integration patterns. This article describes how

the analytics information is retrieved using IBM Integration bus in pattern 4 and Business Process Manager in pattern 5.

Pattern 2 – Analytical Decision Making

This pattern describes how SPSS and Analytical Decision Management can make use of IBM Operational Decision Manager to rank the loyalty of customers based on their order history and then propose loyalty card offers to recommended customers. This shows how decision services can be integrated into SPSS and managed in the same projects as the book order pricing operational decisions.

Pattern 3 – Predictive Scoring Services

This pattern describes how a scoring model can be developed in SPSS and exposed as a Scoring Service which is then used in the operational decision making.

Pattern 4 – IBM Integration Bus Book Order application

In this pattern the book order is supplied as a message in an IIB message flow. The message flow then augments this information with customer 360 degree information including age, gender and loyalty. This information is then used to call the predictive scoring service. All this information is then passed to the BookOrderPricing decision service. On receiving the book order summary from the decision service, the message flow routes the bookOrder message to different queues according to the post and packing price supplied.

Pattern 5 – Business Process Manager Book Order process

In this pattern a simple process is established that allows a customer to enter their book order requirements through a BPM coach. An integration service then augments this information with customer 360 degree information including age, gender and loyalty. This information is then used to call the predictive scoring service. All this information is then passed to the BookOrderPricing decision service and the bookOrder summary response obtained. A decision gateway in the process then checks to see if the post and packing is zero and if so starts an activity to indicate free post and packing. This is simply an alert. The process then uses a coach to display the order and order summary details in all cases.
This pattern shows how to reuse the Book Order Pricing decision service leveraging analytics in the context of a smarter process.

Pattern 1 – Operational Decision Making

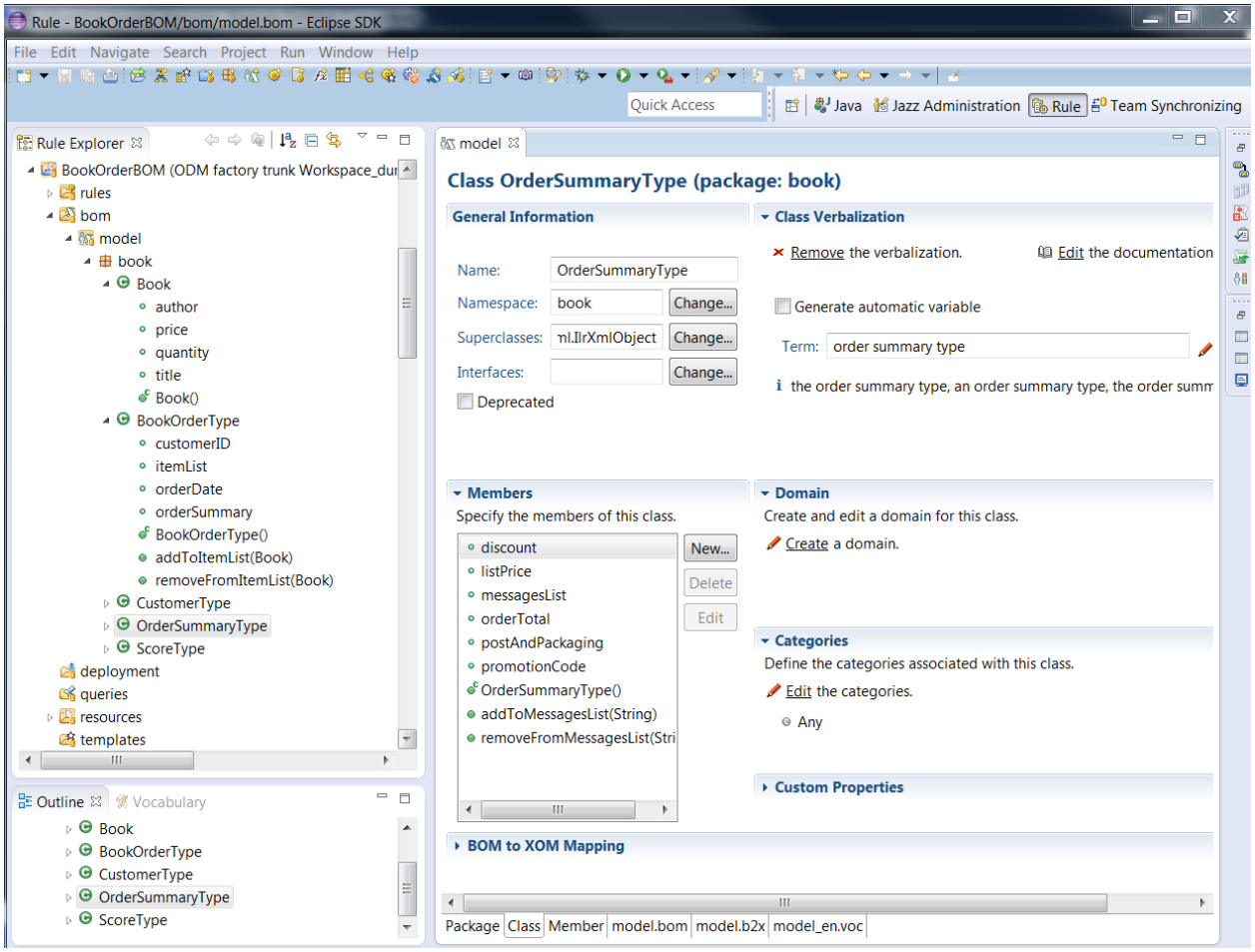
This section describes the new decision service capabilities provided in ODM 8.6 that have been used to realize the scenario decision service that is integrated into the patterns. This groups together the XOM, BOM and all associated rulesets into a single unit of management. Each ruleset is described by a decision operation and is grouped into units of deployment (RuleApps) using a deployment configuration.

When a RuleApp is deployed to Decision Server (or a JSE RES integrated using one of the other patterns), each of the rulesets as defined by their paths is available for execution using the interfaces available on the Decision Server.
For a full Decision Server this includes HTDS (SOAP and REST), EJB, and POJO integration. For the JSE RES integrated into other solutions, the rulesets can be invoked natively realizing the benefits of the pattern.

Book Order BOM and Information Model

The schemas described in the previous section are used to define a Business Object Model as shown below.

Figure 1.1 Book Order BOM



The BOM used for the decision service is based on the schema defined in the previous section. Each object type or class is provided in its own schema file and represented in the BOM.

When designing this BOM and the schema there are a number of points that should be considered.

- ODM does not support arrays of XML objects as parameters. For this reason separate classes should be created in the BOM that reflect these structures (e.g. BookOrderType contains an itemList of Books).

- When refactoring or performing BOM updates, make sure that the package mapping is set-up correctly in the XOM. Using the default may mean that the XOM to BOM mapping will not update correctly and you can end up with complex BOM package hierarchies that will not be able to be integrated into host solutions easily.

- When defining the XOM make sure that all schemas are included in the ruleset archive to allow HTDS WSDL and WADL documents to be able to be generated and the schemas accessed from those descriptor documents. Failure to include the schemas will also mean that the REST "Test" feature will fail. The schemas are also required for IBM Integration Bus to be able to extract the schemas from an imported RuleApp archive.

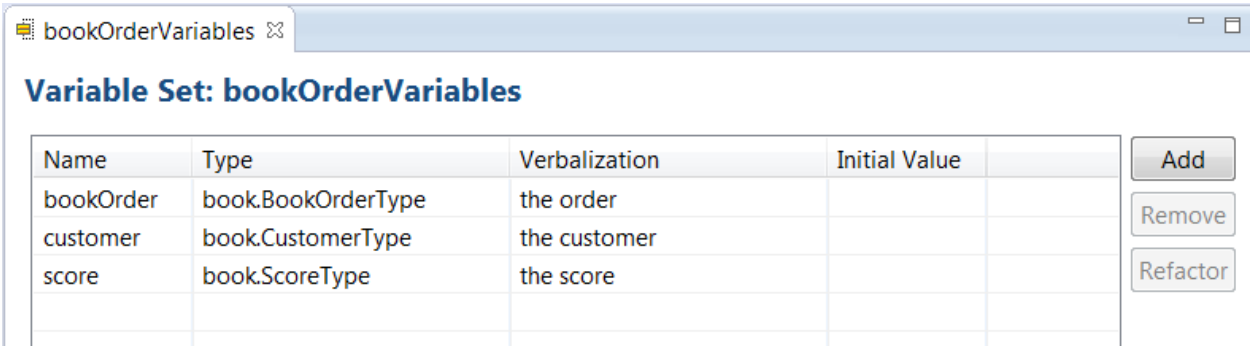
- When obtaining schemas from the ruleset archive, folder hierarchies are not supported. In addition, schema file names are modified and shortened. This means that when using "includes" in a parent schema, the schemas dependencies cannot be resolved. This means that for any given name space or BOM it may be more effective to define a single schema with all the classes within it.

- When defining schemas be sure to be consistent in the use of "form" as qualified or unqualified. Failure to do so will mean that IBM Operational Decision Manager or the host application will not parse the decision service parameters at run time. Unqualified elements and attributes tend to be easier to integrate.

- Be careful with your use of optional elements or nillable elements. Having an uninitialized primitive value (such as **int**) referenced in the rules can often cause a null pointer exception.

Having defined the BOM and its representation of all the terms that the rules will need to consider, we now need to represent the variables that will be used to represent the order.

Figure 1.2 Order Variables



As well as the BOM, the BookOrderDecisionService project hierarchy needs to include variables that are referenced by the rules. These are mapped to the ruleset / operation parameters allowing the book order information to be reasoned with using the rules. In this case the three different sources of information are separated out:

- "the order" – represents the main quote request in terms of the list of books and also holds the quote response with all its discounts applied,
- "the customer" contains all information about the customer that is used for rules,
- "the score" indicates the propensity of the customer to place an order.

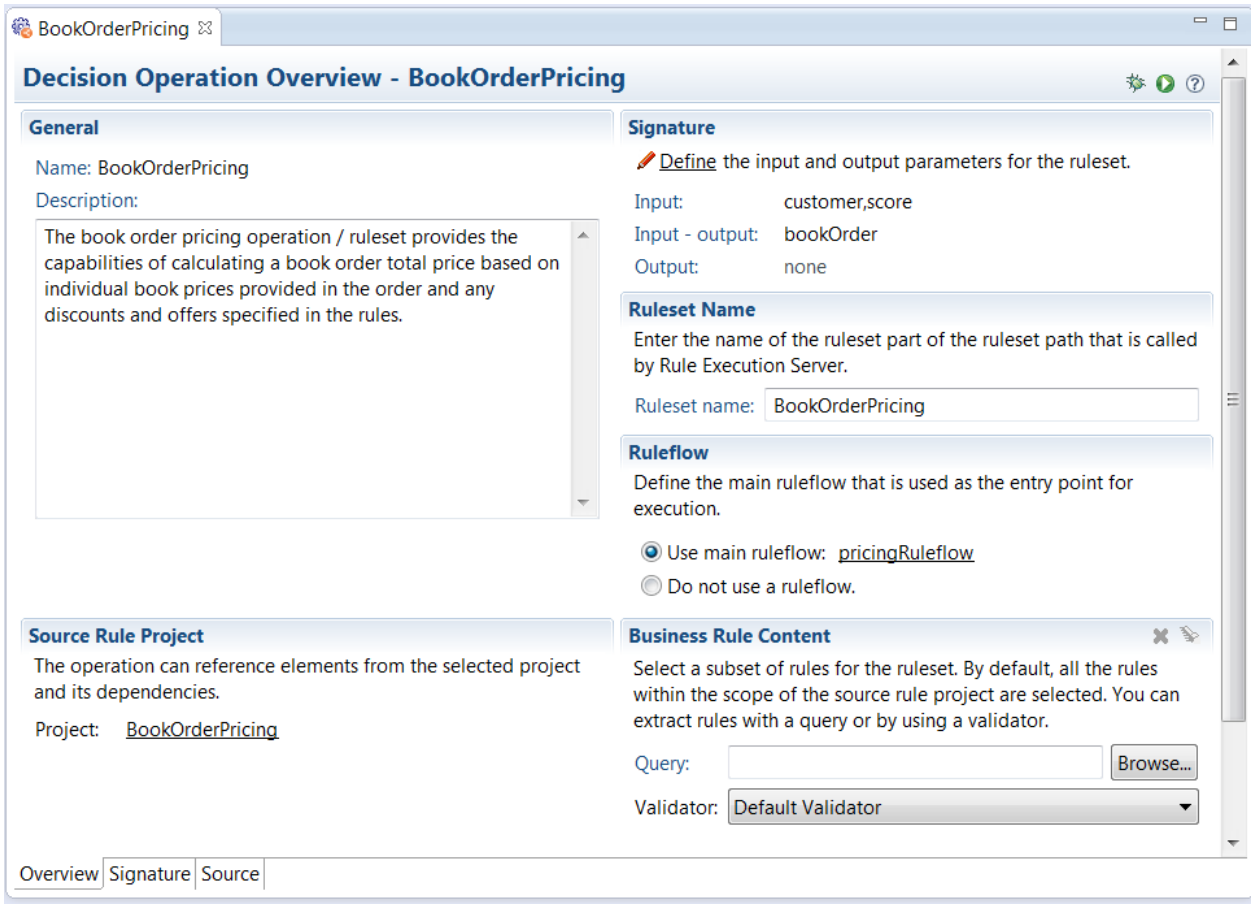
Having defined the variables we can now define the ruleset that will make the pricing decision.

Book Order Pricing ruleset (Operation)

The book order pricing ruleset (or operation) provides rules to determine the discounts and pricing that is required.

The BookOrderPricing ruleset takes the book order, customer and score and determines the price that should be charged for this order taking account of discounts. There is no discrimination about where this information has come from and it will be up to the integration patterns to obtain the necessary fields for the input parameters.

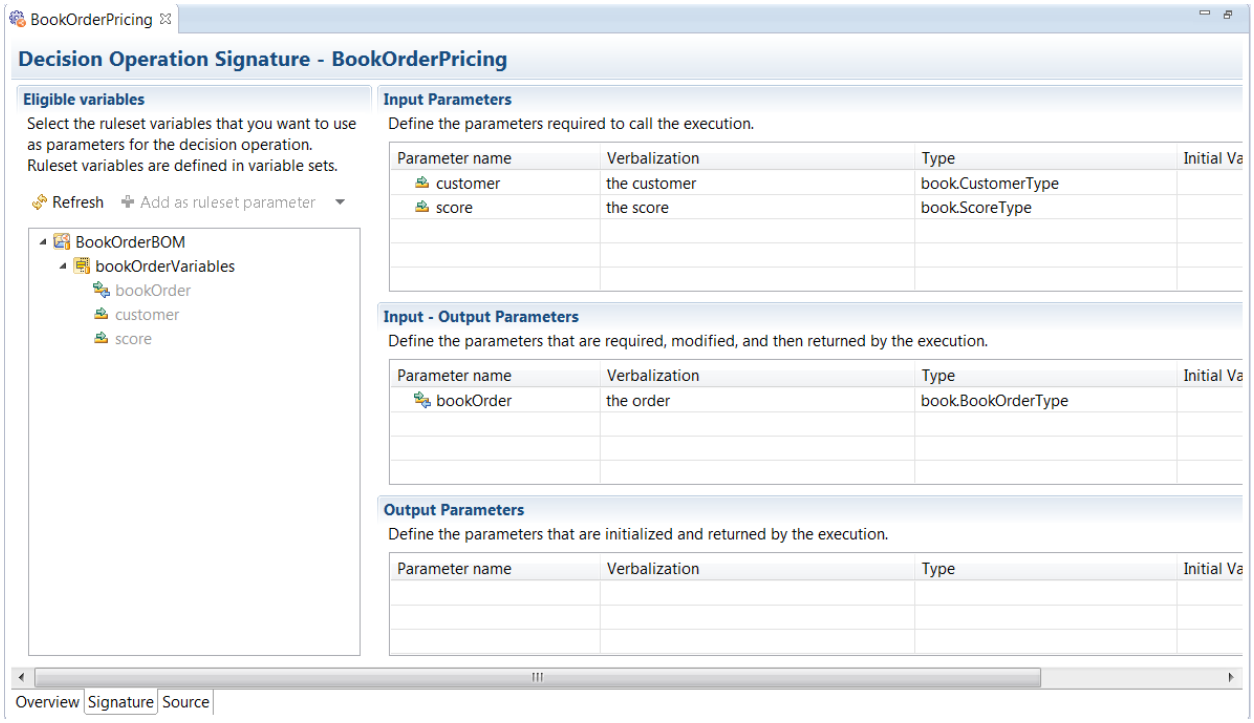
Figure 1.3 BookOrderPricing Decision Operation



The BookOrderPricing operation defines the ruleset that will perform the pricing decision. The rules are all taken from the BookOrderPricing project allowing governance of who can work on or change these pricing rules. The Ruleset Name is used to identify this ruleset within the decision service so that host applications or processes can invoke it consistently with the correct signature and behavior. The operation refers to the rules and their package hierarchy through a ruleflow which determines the order in which rules may be applied. Further options are provided to allow queries or validators to select which rules are actually deployed in the ruleset based on rule meta data.

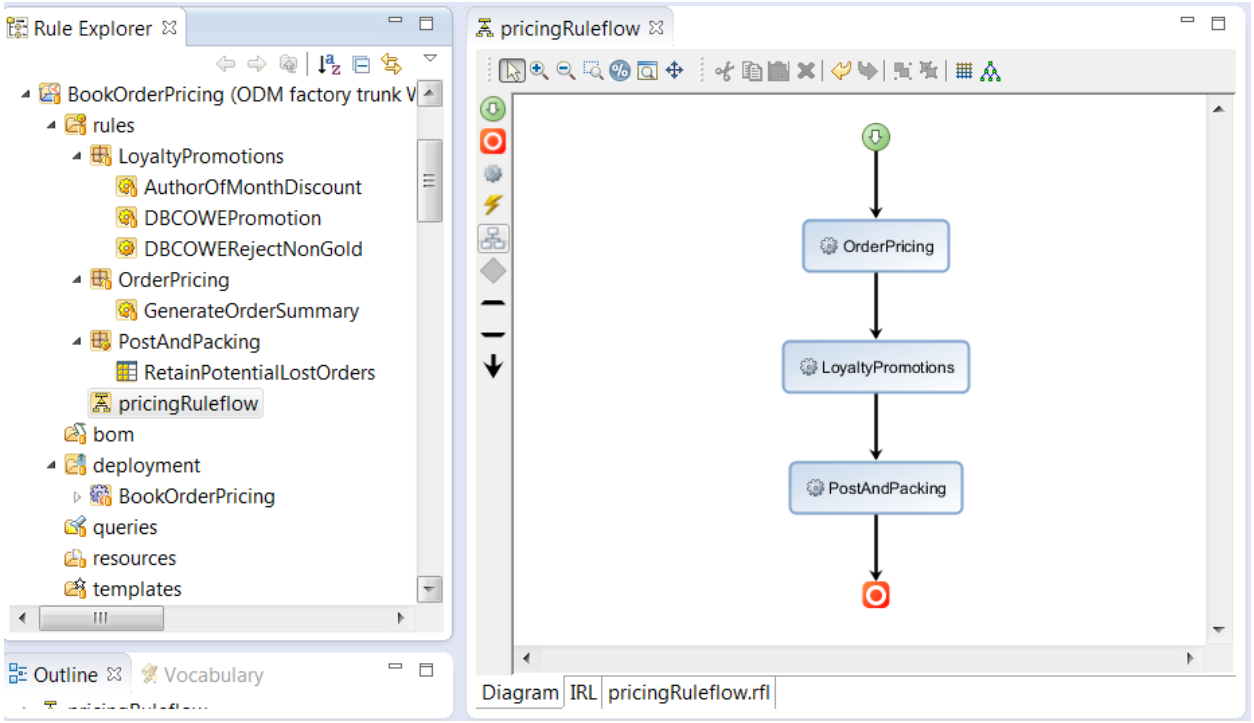
The operation also defines the interface to the ruleset in the form of a signature.

Figure 1.4 Book Order Pricing signature



The signature determines which rule variables are made available in the signature and thus how the rules refer to the order. In this case the bookOrder variable is considered an inout parameter while the customer and score are just used as inputs.

Figure 1.5. Book Order Pricing ruleflow

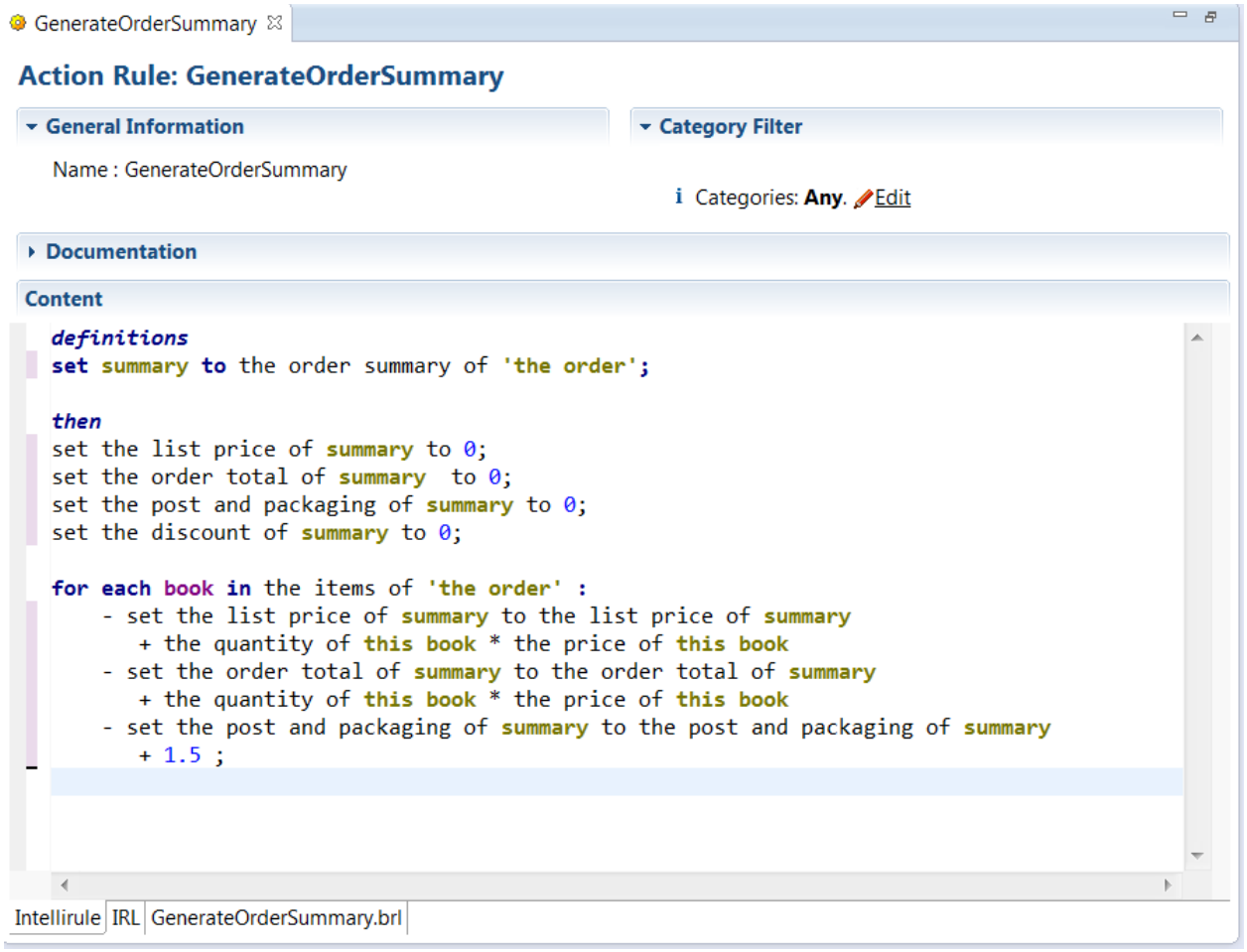


The flow and rules used will evolve over time but consist of three main packages.

- OrderPricing initializes the order summary and calculates the order total price and normal charge for post and packing.
- LoyaltyPromotions calculates discounts and promotions that are applicable for this order based on the order and customer 360 degree information...
- PostAndPacking calculates any post and packing discounts needed to improve the chance of the customer accepting the order.

The rules in these packages are described below.

Figure 1.6 Generate Order Summary



In this rule the order summary totals are set to zero and the basic list prices are calculated by summing up all the items in the order. A fixed price of 1.5 per item is added for post and packing.

Figure 1.7 Author of the month discount

AuthorOfMonthDiscount

Action Rule: AuthorOfMonthDiscount

General Information | Category Filter

Documentation

Content

```
definitions
set summary to the order summary of 'the order';
set authorOfTheMonth to "L P James";
set 'discountValue' to the order total of the order summary of 'the order' * 25 / 100;

if   there are at least 2 books in the items of 'the order'
   where the author of each book contains authorOfTheMonth ,
then
 set the order total of summary to the order total of summary
     - 'discountValue' ;
 set the discount of summary to the discount of summary
     + 'discountValue';
 add "Author of the month discount of 25% applied" to the messages of summary ;
```

Figure 1.8. DBCOWE Promotion

DBCOWEPromotion

Action Rule: DBCOWEPromotion

General Information | Category Filter

Documentation

Content

```
definitions
set summary to the order summary of 'the order';

if the promotion code of summary  is "DBCOWE"
and the loyalty of 'the customer' is "GOLD"
then
  add "DBCOWE Promotion for Gold Customers. One pound off each book."
     to the messages of summary ;
  for each book in the items of 'the order' :
    - set the order total of summary to the order total of summary
        - 1.0
    - set the discount of summary to the discount of summary
        + 1.0
    - add ( "1.00 off: " + the title of this book + " by " + the author of this book )
       to the messages of summary ;
```


This rule first checks for the DBCOWE promotion code and then checks to ensure that the customer has a GOLD loyalty. If these conditions are met, a discount of 1 is applied to every book in the list.

Figure 1.9 DBCOWE Reject Non Gold

DBCOWERejectNonGold

Action Rule: DBCOWERejectNonGold

▸ General Information ▸ Category Filter

▸ Documentation

Content

```
definitions
set summary to the order summary of 'the order';

if the promotion code of summary  is "DBCOWE"
  and the loyalty of 'the customer' is not "GOLD"
then
  set the promotion code of summary to "NONE" ;
  add "DBCOWE promotion is only for Gold customers." to the messages of summary ;
```

This rule deactivates the DBCOWE promotion for non-Gold customers and adds a notification message.

Figure 1.10 Retain Potential Lost Orders

RetainPotentialLostOrders

	Order prediction	Min Confidence	Free P&P	Message
1	N	0.6	-	Free special delivery post and packing.
2		*Otherwise*	-	Free post and packing.
3	*Otherwise*		-	No free post and packing.
4			-	
5			-	
6			-	
7			-	
8			-	
9			-	

General | Decision Table | IRL | RetainPotentialLostOrders.dta

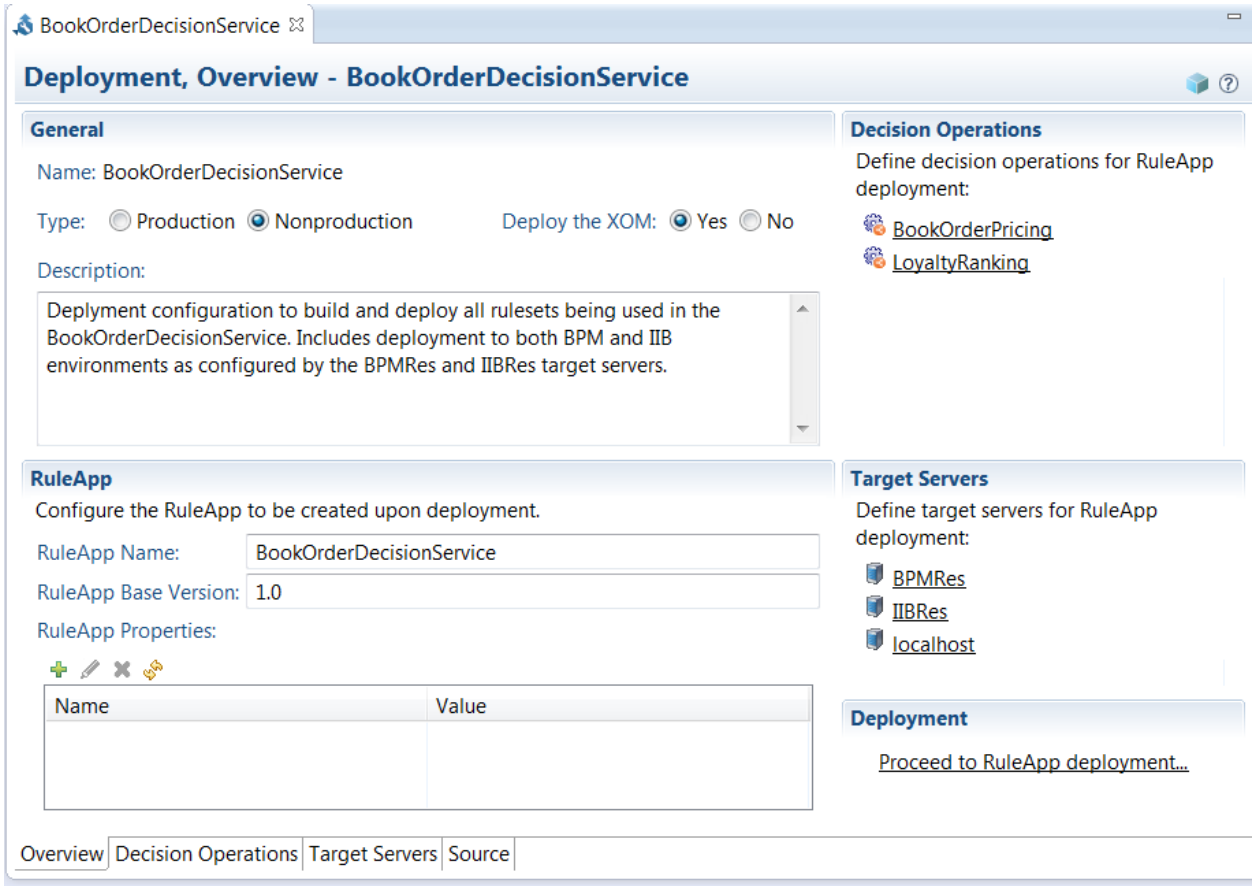
This decision table uses the order placed prediction and confidence provided from the scoring service to offer a good deal on post and packing when it is likely that the customer will reject the order. For situations where there is a high confidence that the order will be rejected a special

delivery free post and packing is offered. For customers that are likely to accept the order, no free post and packing is applied.

Book Order Decision Service Deployment

The new decision service capability allows the decision service project hierarchy to manage and deploy multiple rulesets. The rulesets / operations to be deployed can be configured in a deployment configuration together with their target Rule Execution Servers.

Figure 1.11 Book Order Decision Service deployment configuration

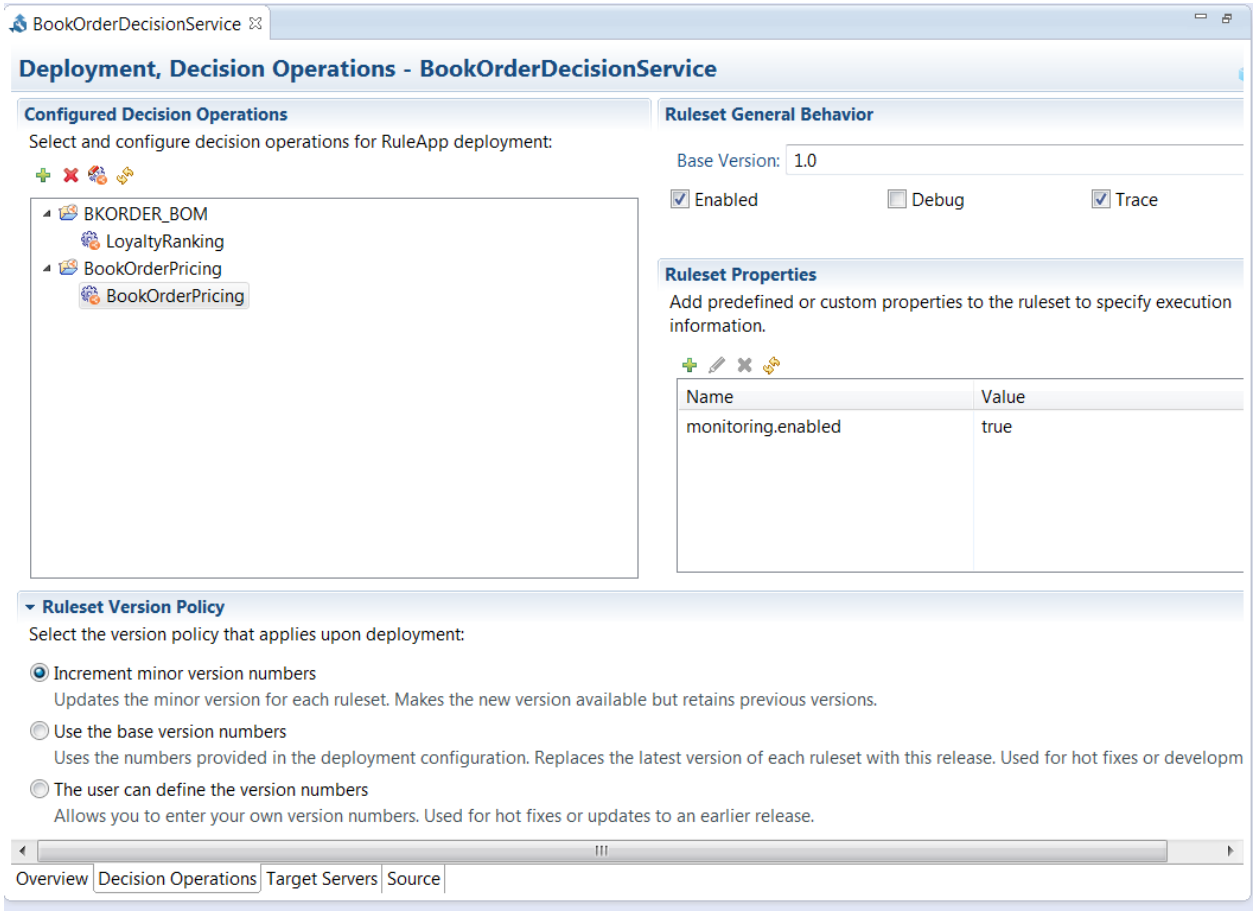


The BookOrderDecisionService deployment configuration provides the container for consistently deploying rulesets to a Decision Server or JSE Rule Execution Server.

In this case the BookOrderPricing operation (and LoyaltyRanking operation used by analytics) are included and are deployed to the BPMRes and IIBRes servers which can be configured to reference the Rule Execution Server consoles that are supporting the integration scenarios. This deployment ensures that whenever the rules are updated, they are deployed consistently across the different environments and servers.

The decision operations tab shows how each ruleset is to be deployed.

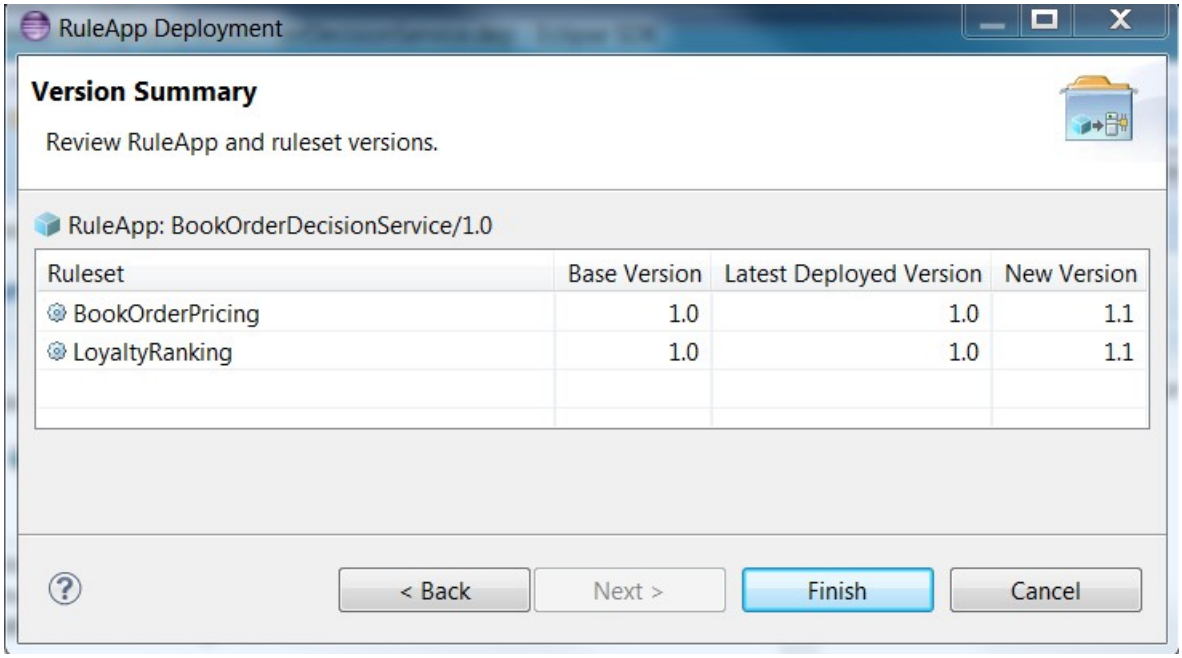
Figure 1.12 Operation Deployment Configuration



The operations tab allows properties of the ruleset to be configured which affects how it executes at runtime. In this case the BookOrderPricing is configured to allow tracing and monitoring in the decision warehouse.
The ruleset version policy can also be defined here. This is normally set as Increment minor version numbers which will create a new minor version for each ruleset deployed. If an application is always looking for the latest version of a ruleset, this means that each deployment will override previous versions. In the case of a problem the latest deployment can be deleted and the host application will automatically revert to its predecessor.

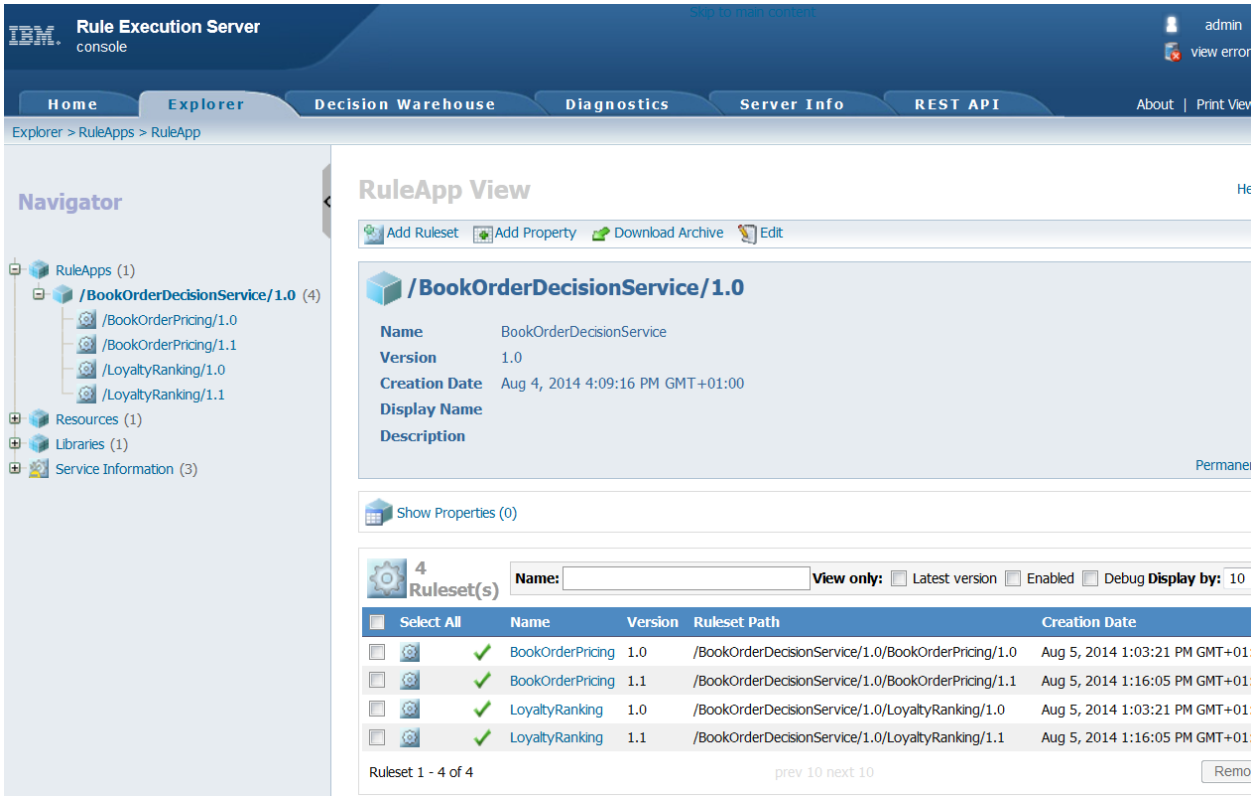
When deploying using a deployment configuration the deployment reports show the versions present and the version to be used on this deployment.

Figure 1.13. Deployment Versioning



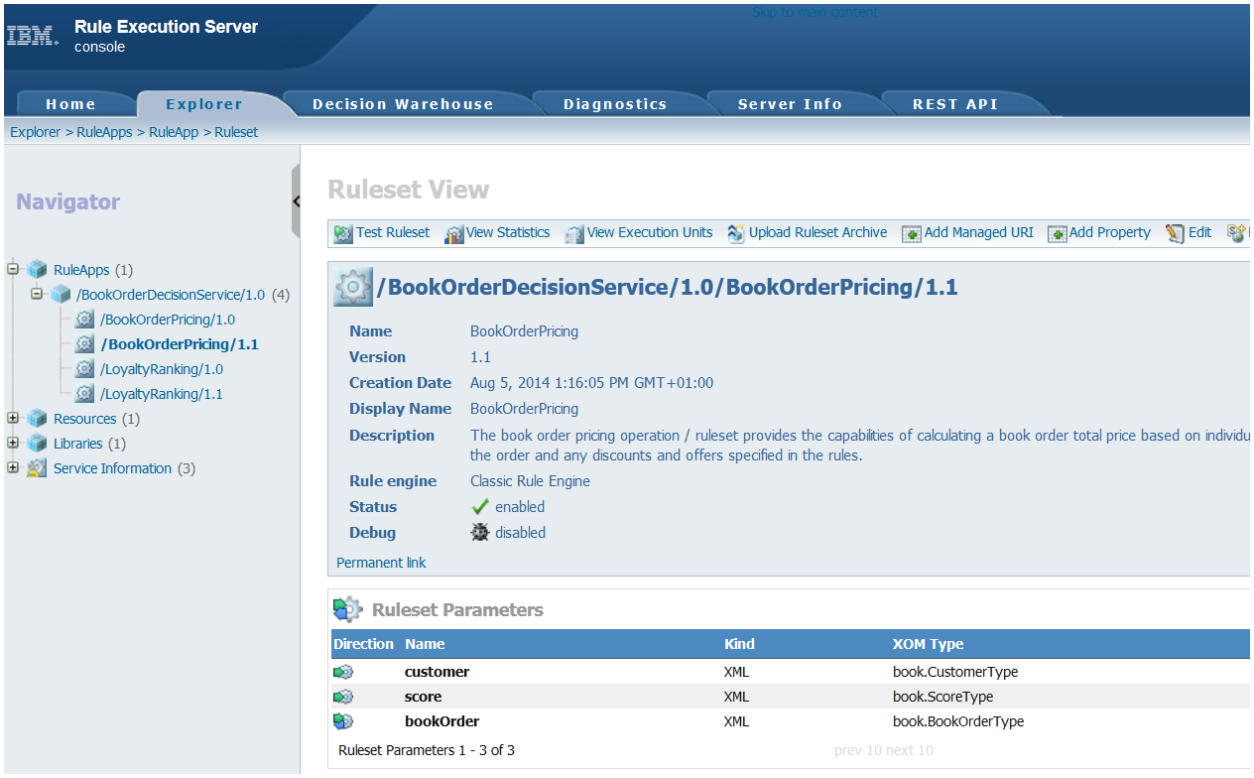
Following deployment, the decision service and its rulesets can be seen in the target Rule Execution Servers.

Figure 1.14 Rule Execution Server Console showing deployed Decision Service



From the Rule Execution Server console explorer tab the details of the rulesets can be seen for testing and obtaining Web Service interfaces (WSDL and REST).

Figure 1.15. Rule Execution Server Console Ruleset View



This view can be used to test the rulesets, view statistics on ruleset execution and obtain WSDL and WADL definitions for HTDS web services to be integrated into other solutions.

Pattern 2 – Analytical Decision Making

This pattern describes how rules and decision services can be invoked from within SPSS as part of Analytical Decision Management applications.
In the scenario we start this pattern with information that has been built up over time from customers buying books. This information is stored in a database and at regular intervals the Analytical Decision Management application can examine the database and recommend customers that should have loyalty upgrades. As part of this scenario the pattern shows how ADM can invoke external ODM rules to recommend the loyalty for the customer. It is not the intention to show how ADM could implement the book order pricing scenario although readers with experience of Analytical Decision Management should be able to use this pattern when leveraging ODM in their solutions.

Analytics Information Model

SPSS processes information and applies analytic functions using streams. These streams take information from a variety of sources and produce outputs that can be used in other streams or decision making applications.

This scenario takes information from a BKORDER database with three tables defined in a schema named ADMINISTRATOR.

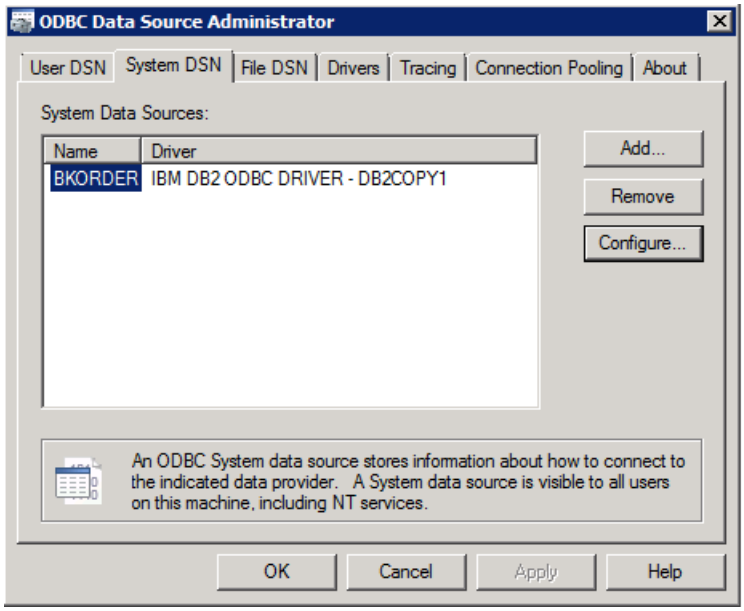
- CUSTOMER – containing 360 degree information about the customer
- ORDER – containing past order summary information
- BOOK – containing information about the books being sold

The database is populated with some sample data that can be used to show the principles of combining analytics and operational decision making. In reality this database would be populated from the results of the order pricing and the customer's acceptance of the orders.

The database needs to be exposed as an ODBC source in order for SPSS Modeler Server to gain access to it. The IBM SPSS Data Access Pack (SDAP) provides ODBC drivers from DataDirect that are for exclusive use with IBM SPSS applications for all platforms. See IBM Business Analytics Proven Practices: IBM SPSS Modeler - ODBC Configuration Best Practices and Troubleshooting for configuration details.

For this scenario, the SDAP is not installed on the windows platforms and the database is registered using the ODBC administration tool as shown in figure 2.1.

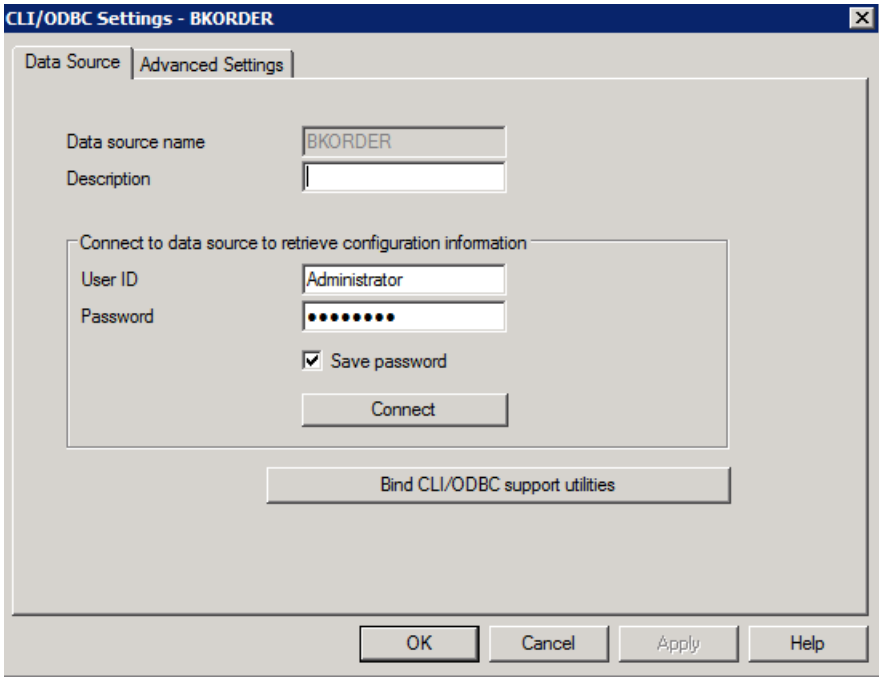
Figure 2.1 BKORDER ODBC data source



This source can then be discovered by the SPSS Modeler Server or SPSS Modeler client. Note that ODBC data sources must be declared locally everywhere they are referenced. This means if you are running a Modeler stream on a local laptop client, you will need to configure the ODBC source on that laptop.

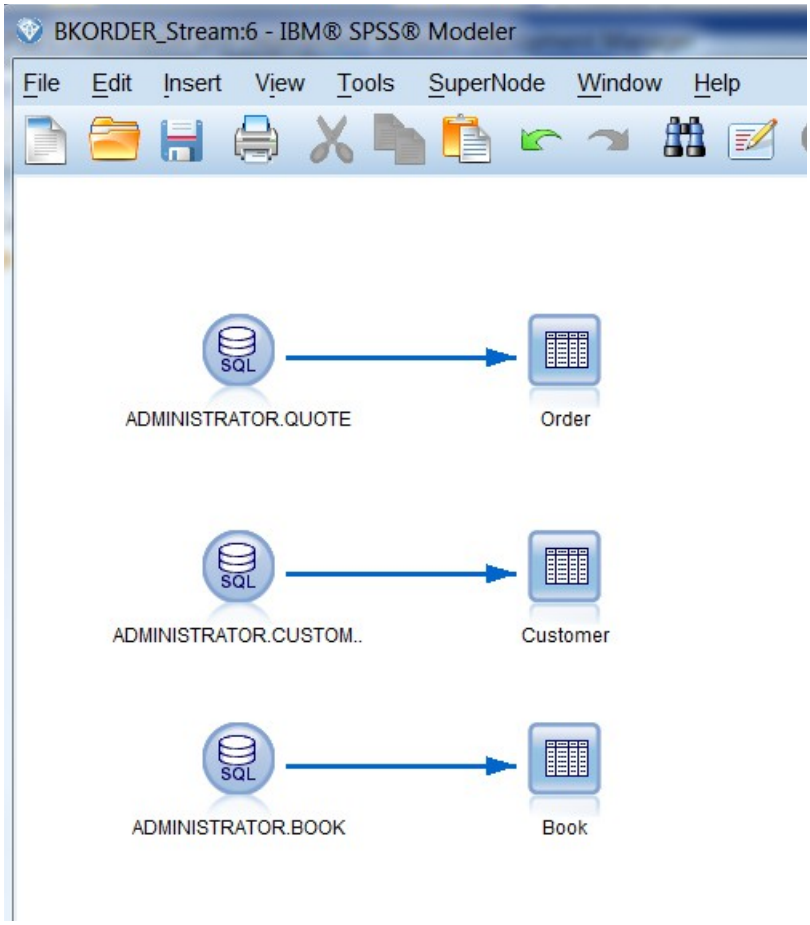
Care should be taken to ensure that users have the right credentials. The ODBC source can be configured to store credentials and thus ease access to the database.

Figure 2.2 BKORDER ODBC data source credentials



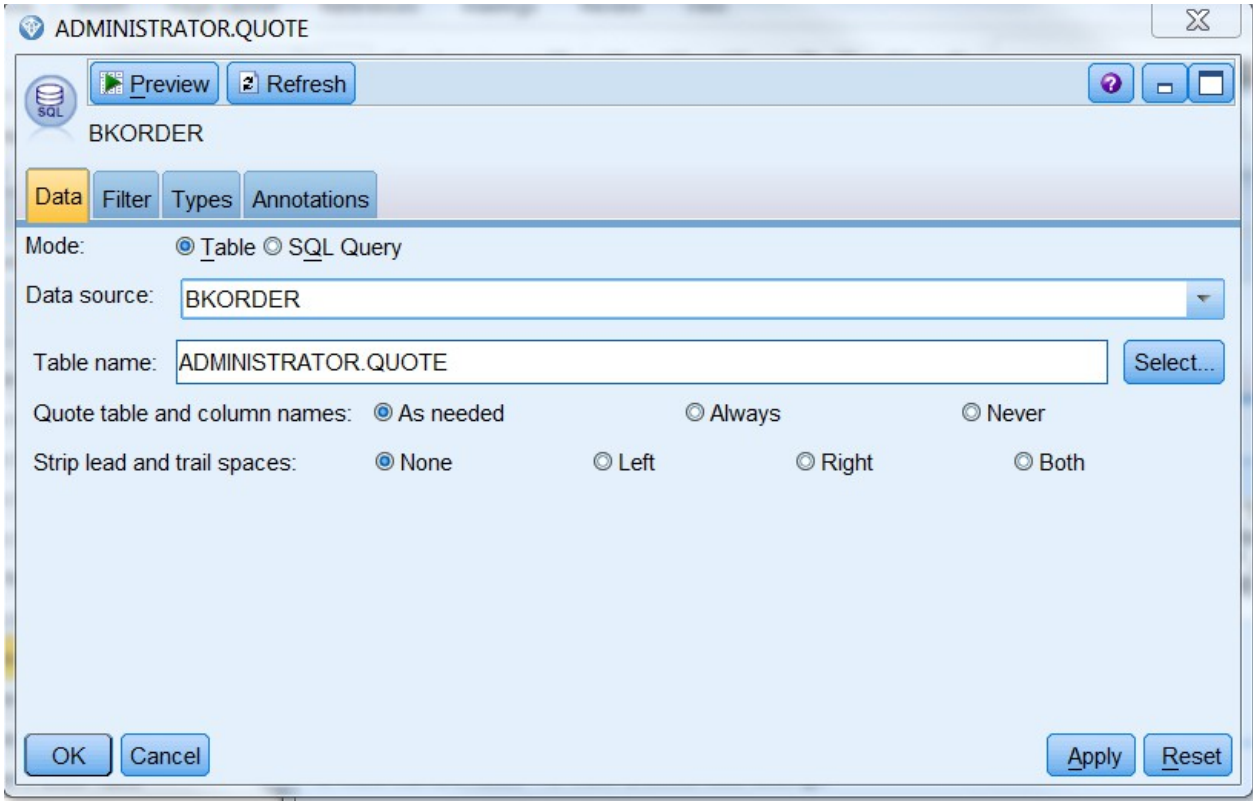
The Analytics processing scenario then provides a single stream to hold these tables as show below.

Figure 2.3 BKORDER_Stream



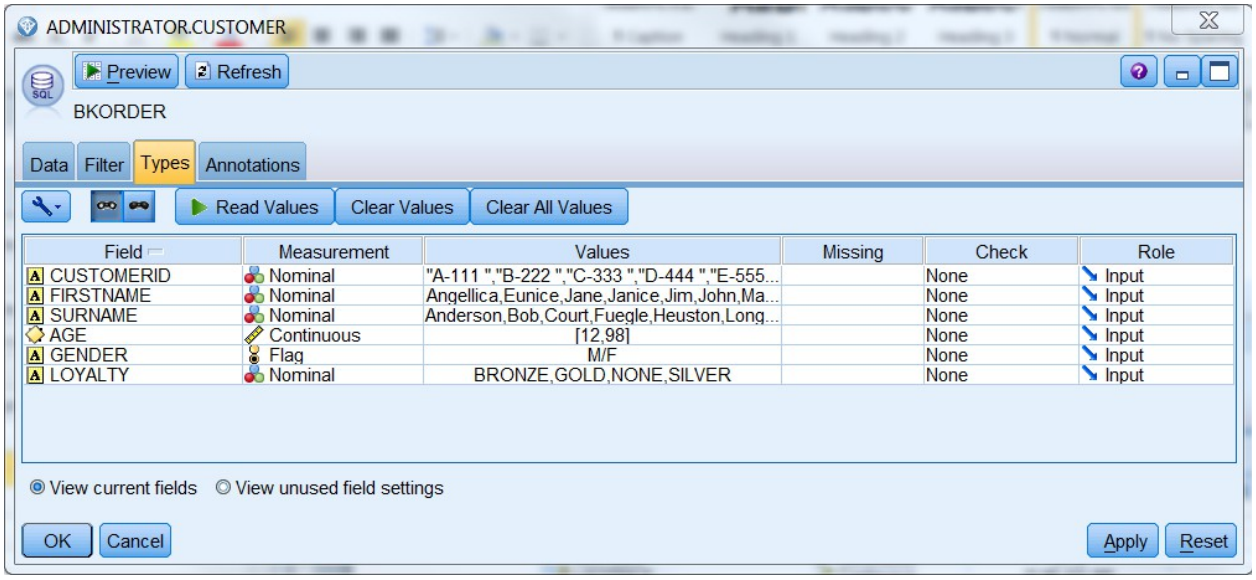
Each table is configured individually as a data source and a table output. Other processing nodes can be added to the stream to refine or filter the raw database information. Each table needs to be connected to an ODBC data source as shown below.

Figure 2.4. BKORDER Datasource configuration



Each table provides a number of fields and can be referenced independently. The fields and content of each table are shown below

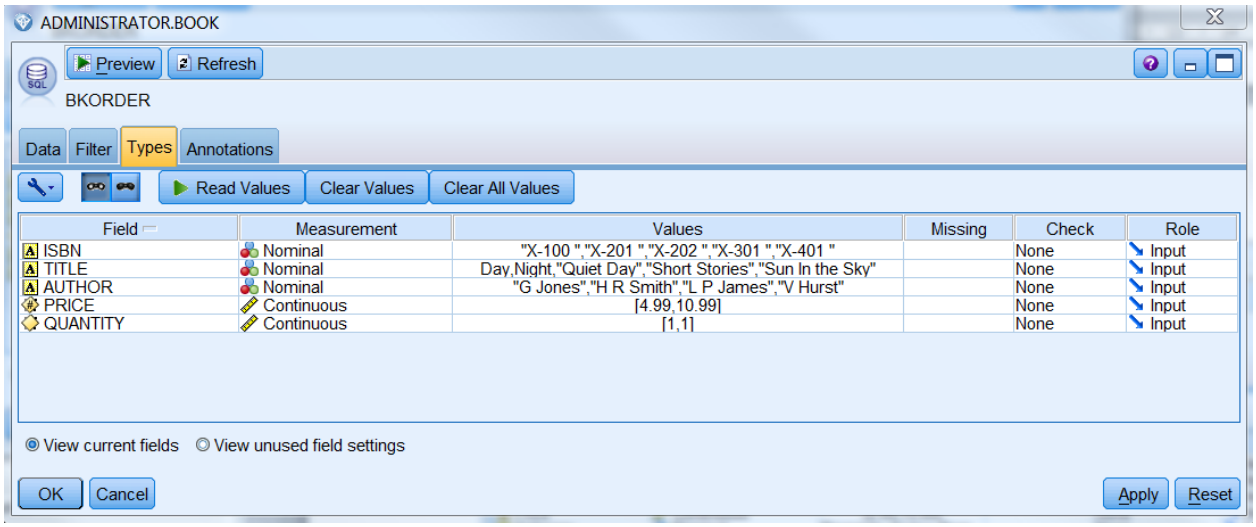
Figure 2.5 Customer Table



The customer table provides the key 360 degree information about the customer.

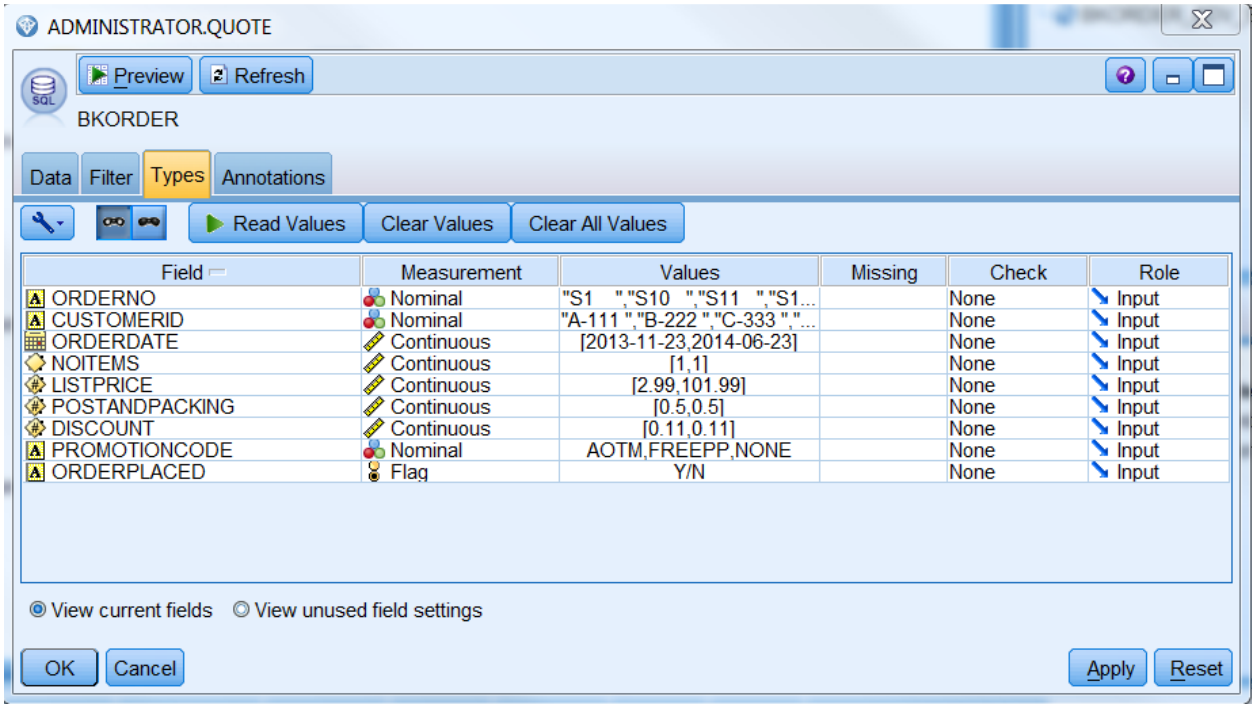
- CUSTOMERID is a unique key used to relate orders to customers.
- FIRSTNAME and SURNAME are the names of the customer.
- AGE denotes the age in years of the customer.
- GENDER denotes the gender of the customer as 'M' or 'F'.
- LOYALTY denotes the Loyalty Card held by the customer as Bronze, Silver or Gold. This field is maintained for each customer using a combination of analytics and business rules.

Figure 2.6 Book Table



The ISBN field provides a unique key that can be used obtain information about a particular book. For this scenario the important information would be the PRICE of the book but other fields such as AUTHOR or QUANTITY may be used to make discount decisions about an order. This table is not currently used in the tutorial.

Figure 2.7 Quote Table



The Quote table provides information about the quotes that have been provided to customers and whether they have been accepted or not.

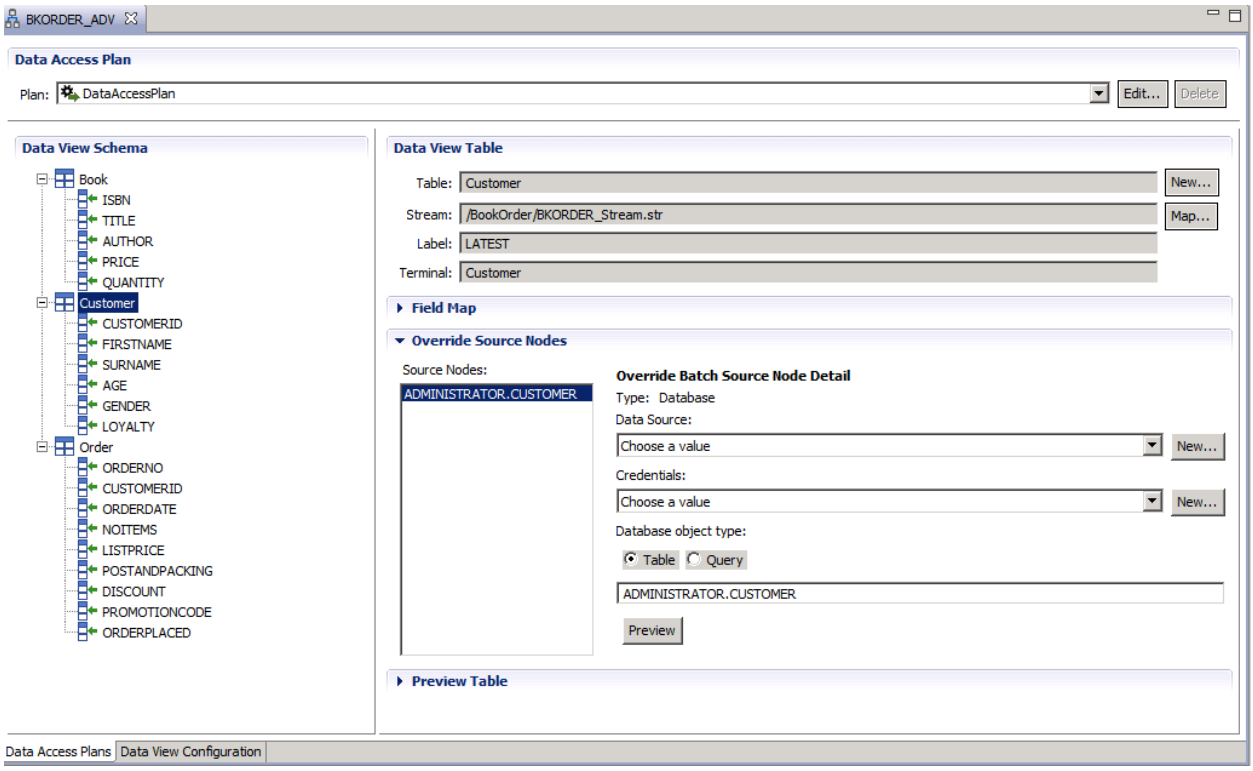
- CUSTOMERID provides a link to customer details that may be useful in making decisions about the customer.
- ORDERDATE allows temporal filtering of orders to be undertaken in rules or analytical processing.
- NOITEMS is the number of books in the order (either a list of multiple books or a repeated single book order).
- LISTPRICE is the total list price excluding discounts and postage and packing.
- POSTANDPACKING indicates the amount charged for post and packing on this quote.
- DISCOUNT defines the total discount applied to the order. The price is effectively the list price less the discount.
- PROMOTIONCODE is used to indicate any promotions applied to this order.
- ORDERPLACED indicates if the order was placed (Y) or if the quote supplied was not accepted (N).

Analytic Data View

To build up a 360 degree view of a customer and their value to the book ordering organization, an Analytic Data View can be created. This allows the stream outputs to be combined in a single view of the customer through the addition of hierarchical and aggregate fields.

An Analytic Data View must be defined using the C&DS Deployment Manager as show for BKORDER_ADV below. This is based on the BKORDER_Stream tables defined earlier

Figure 2.8 BKORDER_ADV



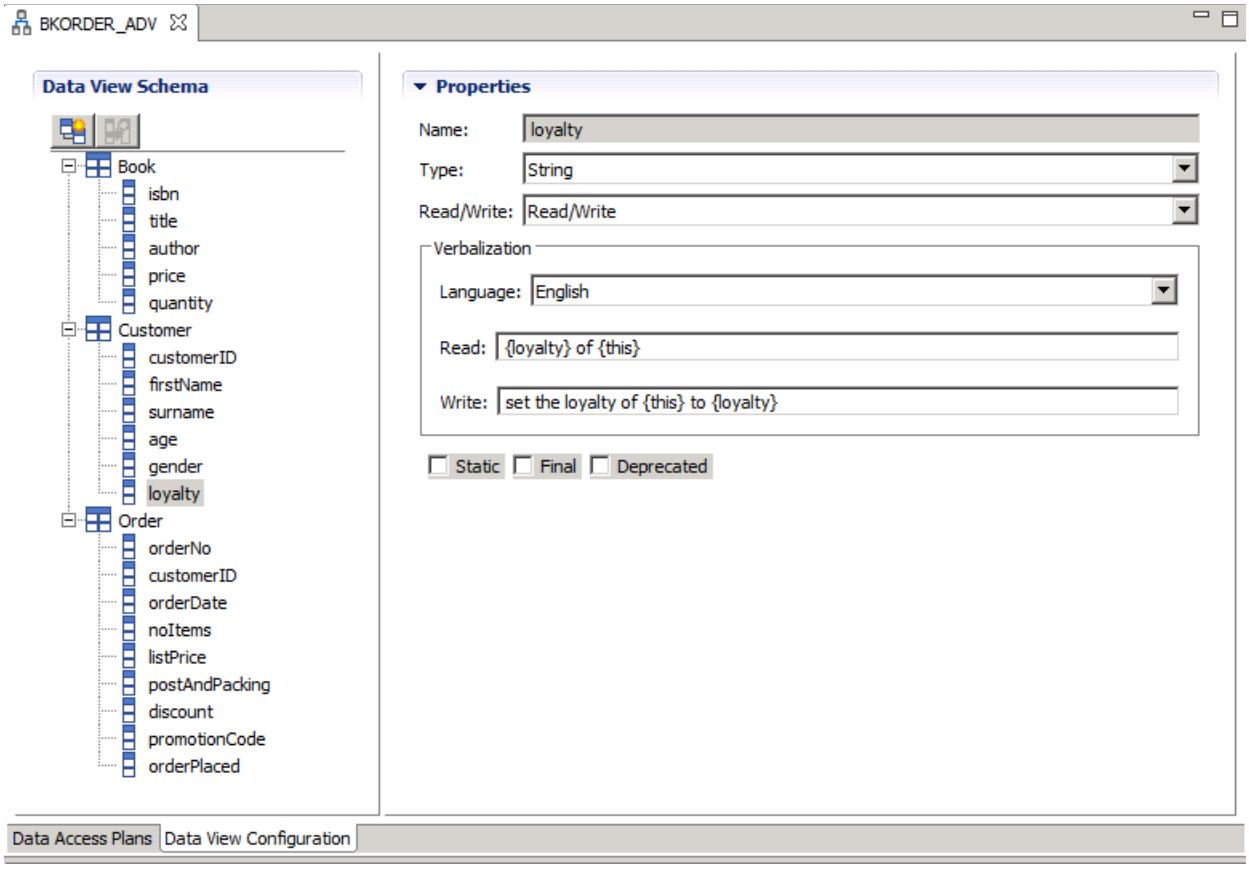
This ADV combines three tables from the BKORDER stream:

- Book - containing details about an individual book
- Customer – containing the customer details from the database
- Order – containing book quotes related to each customer.

The fields used in the ADV are initially taken from the stream output but may be customized in the ADV to make them more understandable and less ambiguous.

In this case the capitalized values have been renamed to provide more human readable terms when used in the stream processing or rules.

Figure 2.9 BKORDER_ADV field name customization



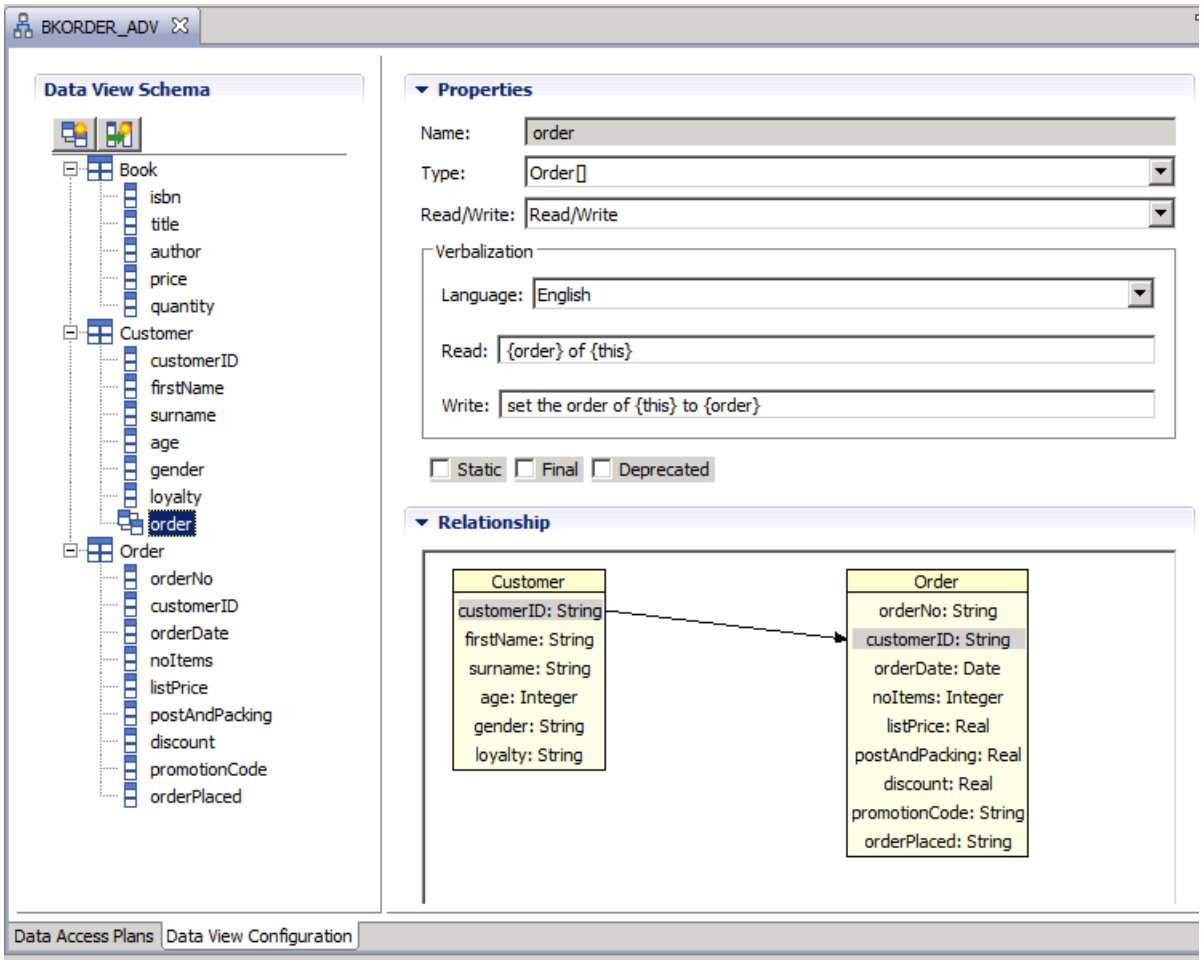
The Data View Configuration tab allows the representation of each field in the schema to be defined (the XOM) and also determines the representation of each field when used in rules.

ADVs provide the ability to add two new types of attribute

- Collection Attributes – allow a hierarchical structure to be represented
- Derived Attributes – allow rules to be defined that indicate how an aggregate value can be added to the view based on the set of attributes in a collection. Derived attributes are not included in the XOM or BOM and are not therefore passed to ODM to be used in making decisions.

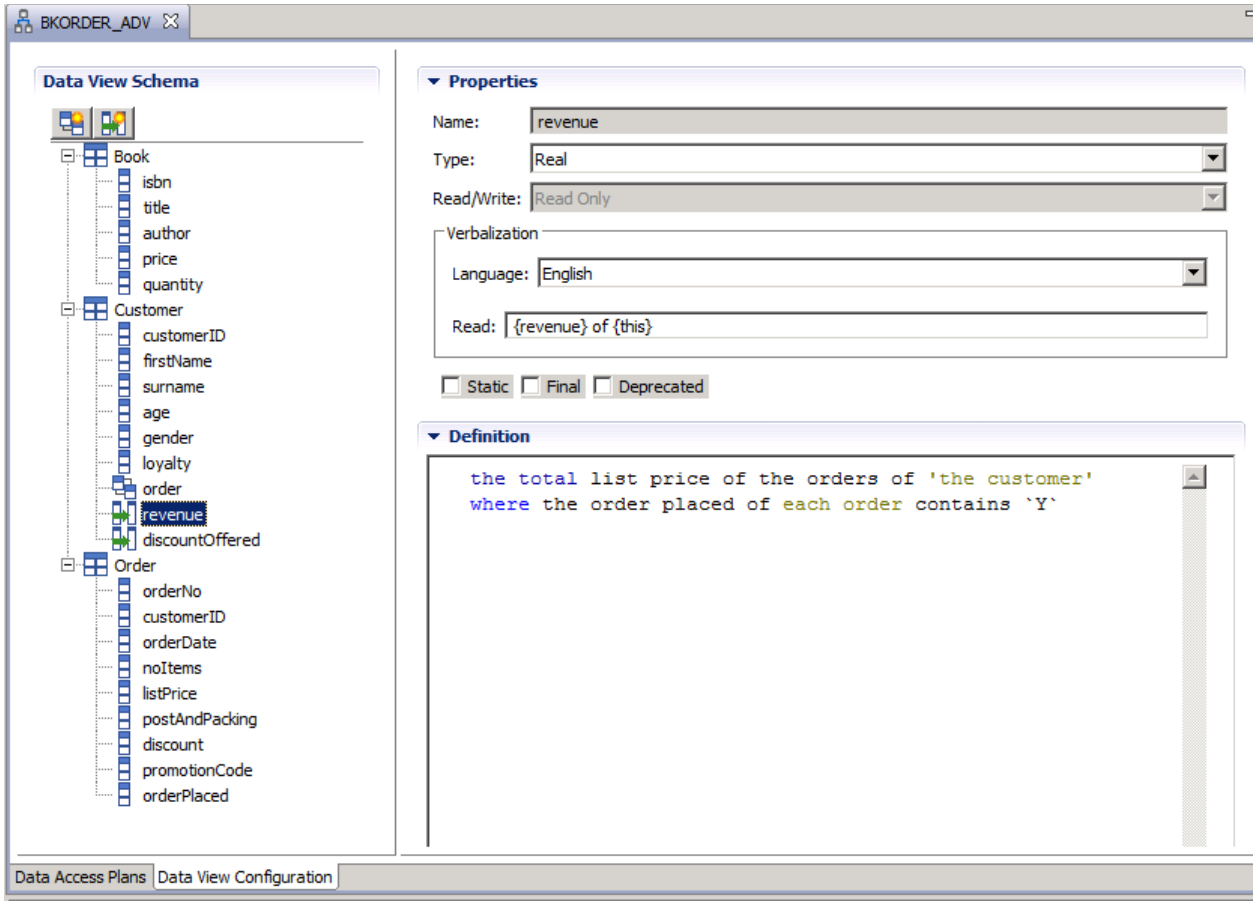
Each of the attributes of Analytic Data View tables can have a verbalization defined for them. This allows the rules in the derived attributes to refer to each attribute using natural language. This verbalization can be provided for different locales and depends on whether the variables can be read, written or both. The verbalization is held in the Business Object Model. Both the XOM and the BOM models can be exported and used in ODM so that the same language is used in both the analytic models and the business rules.

Figure 2.10 BKORDER_ADV Order Collection Attribute



For this scenario we have created an "order" collection attribute of Orders in the Customer. To associate each order with the correct customer, you need to drag the Customer.customerID key to the equivalent Order.customerID key. With this relationship, the Customer.order collection will contain all orders provided to this particular customer.

Figure 2.11 BKORDER_ADV derived attributes



The revenue derived attribute uses business rules to calculate the value of orders from this particular customer. All derived attributes are read only and may be verbalized (and thus used in other derived attributes) in the same way as other data view attributes.

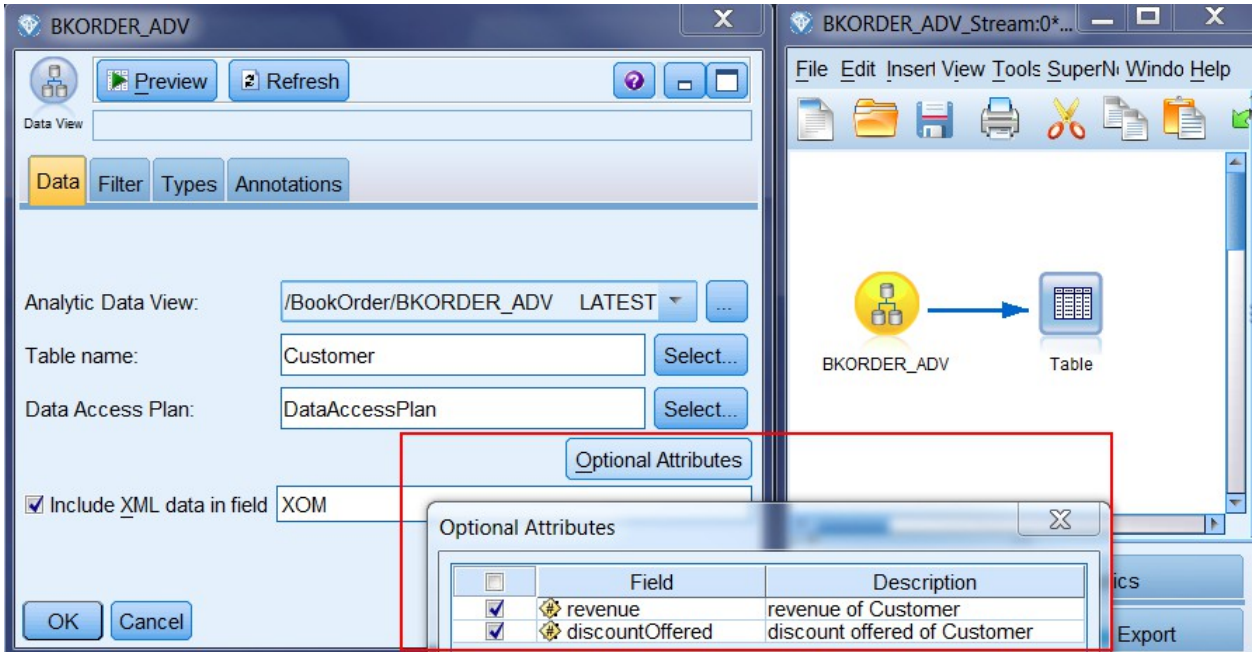
In this case the rule uses an aggregate operator, “the total” which sums the price field from any order where the customer ID matches that of the customer. The “where” clause adds conditions that restrict the summation to those orders for which an order was placed.

The discountOffered derived attribute follows a similar pattern. The attribute simply sums the discounts offered to the customer irrespective of whether the customer accepted the order or not.

As users type rules, the rule editor prompts for terms from the vocabulary defined for this ADV. Errors in the rules are also shown allowing quick resolution of terms.

Once an ADV has been defined in C&DS Deployment manager it can be used in a Modeler stream and executed as shown below in figure 2.12.

Figure 2.12 BKORDER_ADV stream



A "Data View Source" node is dragged onto the canvas, together with a table output and the two nodes are connected up. The Data View Source can then be configured to reference the ADV defined in C&DS Deployment manager as shown in figure 2.12.

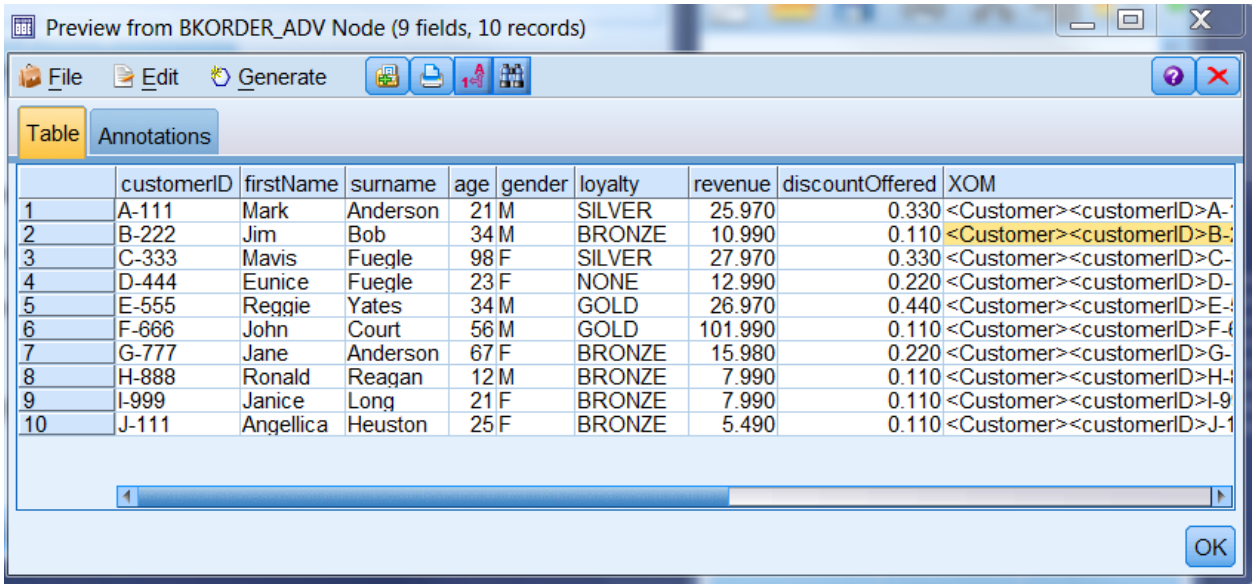
When configuring the Data View source, the Analytic Data View is first selected. This ADV should exist in the C&DS repository from where the analysis will be run. You then select the table to use as a source (Customer) and then the Data Access Plan which in this case is the default DataAccessPlan plan.

The derived attributes are optionally included in the source and may be selected by clicking the **Optional Attributes** button.

Selecting the **Include XML data in field** box will allow the full hierarchy of the ADV to be seen as an XML document. This is the information that will be passed to ODM when making a decision but is not used internally in SPSS.

Once the Data View source has been configured, the ADV can be executed by invoking **Preview** on the data view source. The Preview view shows the ADV fields and their values for each customer in the stream.

Figure 2.13 BKORDER_ADV Preview



	customerID	firstName	surname	age	gender	loyalty	revenue	discountOffered	XOM
1	A-111	Mark	Anderson	21	M	SILVER	25.970	0.330	<Customer><customerID>A-
2	B-222	Jim	Bob	34	M	BRONZE	10.990	0.110	<Customer><customerID>B-
3	C-333	Mavis	Fuegle	98	F	SILVER	27.970	0.330	<Customer><customerID>C-
4	D-444	Eunice	Fuegle	23	F	NONE	12.990	0.220	<Customer><customerID>D-
5	E-555	Reggie	Yates	34	M	GOLD	26.970	0.440	<Customer><customerID>E-
6	F-666	John	Court	56	M	GOLD	101.990	0.110	<Customer><customerID>F-
7	G-777	Jane	Anderson	67	F	BRONZE	15.980	0.220	<Customer><customerID>G-
8	H-888	Ronald	Reagan	12	M	BRONZE	7.990	0.110	<Customer><customerID>H-
9	I-999	Janice	Long	21	F	BRONZE	7.990	0.110	<Customer><customerID>I-9
10	J-111	Angellica	Heuston	25	F	BRONZE	5.490	0.110	<Customer><customerID>J-1

This information can then be used in other streams and analytics processing to make decisions about discounts or loyalty cards to offer a customer.

Creating an ODM Decision service for use in SPSS

In many analytic applications it is useful to be able to use rules and decision services as part of the analytics for processes such as classification. The first step to integrating ODM into SPSS processing is to ensure that the information available in the modeler streams is available to the ODM ruleset or decision service and that the decisions provided by the rules can be used in the analytics processing. There are two approaches for defining these shared information models:

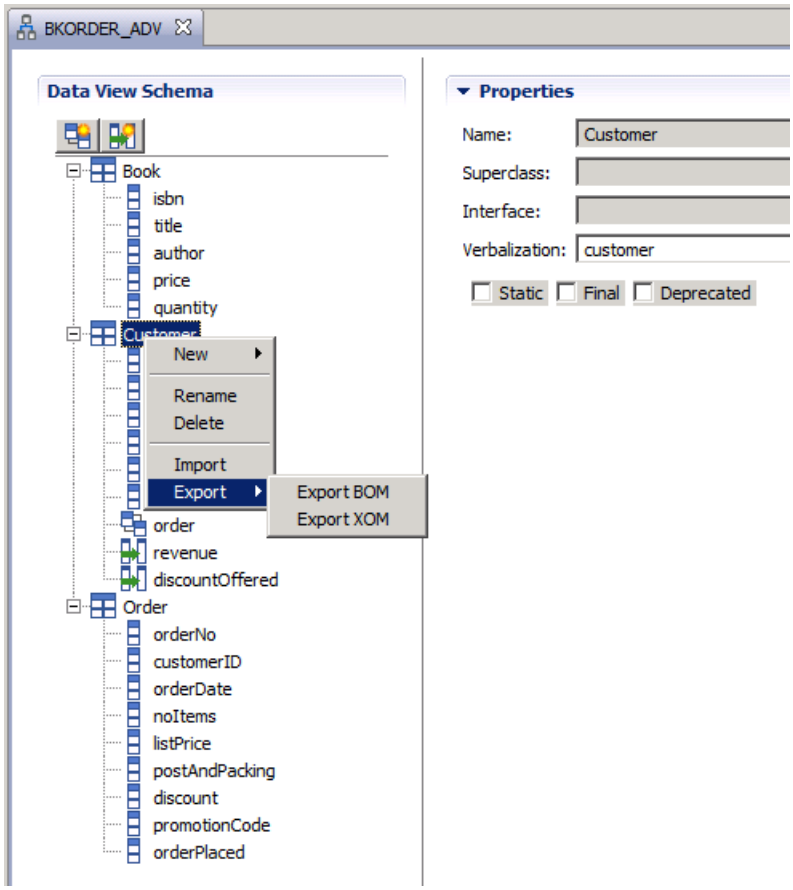
- Using an Analytical data view – this allows hierarchical information to be passed to the decision and allows ADM applications to use information returned back from ODM
- Using a flat model – this is defined based on meta data exported from an Analytical Decision Management project.

This tutorial describes using the Analytical Data View that we have already configured in the project. In this scenario we will create a decision service that uses the BOM provided by SPSS and contains rules for classifying customer loyalty into Gold, Silver or Bronze as well as using the model for other operational rules.

The ADV can export a schema representing the information passed to ODM at runtime as well as a Business Object Model that represents the vocabulary defined in the ADV. This means that the same language and terminology can be used in both the SPSS derived attributes and the ODM rules.

To export these artifacts open C&DS Deployment manager and select the BKORDER_ADV as shown in figure 2.14.

Figure 2.14 Exporting Object Models from an Analytical Data View



To export the Object Models perform the following steps:

1. Select a table
2. right–click Export > BOM to export the BOM as BKORDER_BOM.zip
3. right–click Export > XOM to export the BOM as BKORDER_XOM.zip

Due to a defect in C&DS 6.0.0.1, you may find that the BOM will not export successfully. This is a known defect and will be resolved in the next fix pack.

These should be saved and made available to import into Rule Designer.

In Rule Designer create a new decision service BOM project called BKORDER_Loyalty.

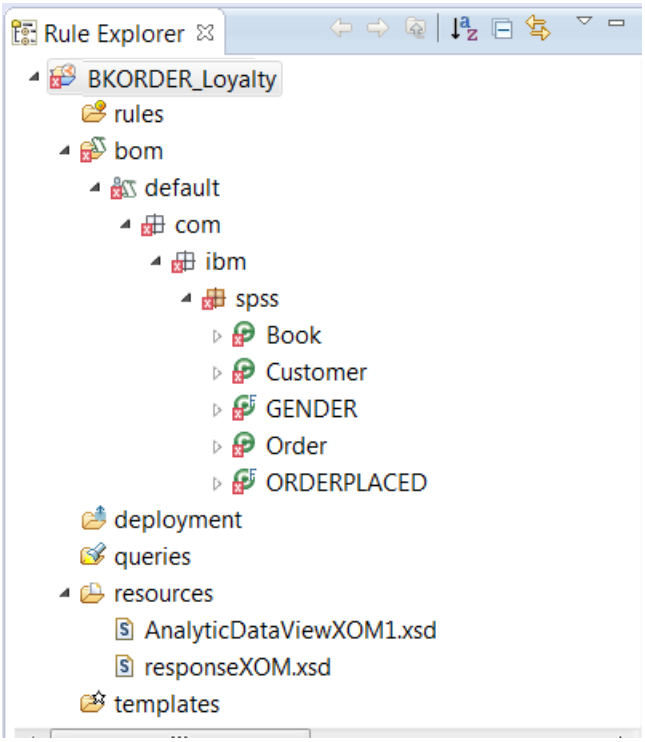
Import the SPSS artifacts into your Rule Designer workspace using the following steps

1. File > Import > Archive File
2. Browse to select BKORDER_XOM.zip
3. Select project folder: BKORDER_Loyalty/resources
4. Select Finish
5. File > Import > Archive File

6. Select project folder: BKORDER_Loyalty/bom
7. Select Finish

After the import the artifacts are visible in the Rule Project but are in error as they are not yet integrated into the project.

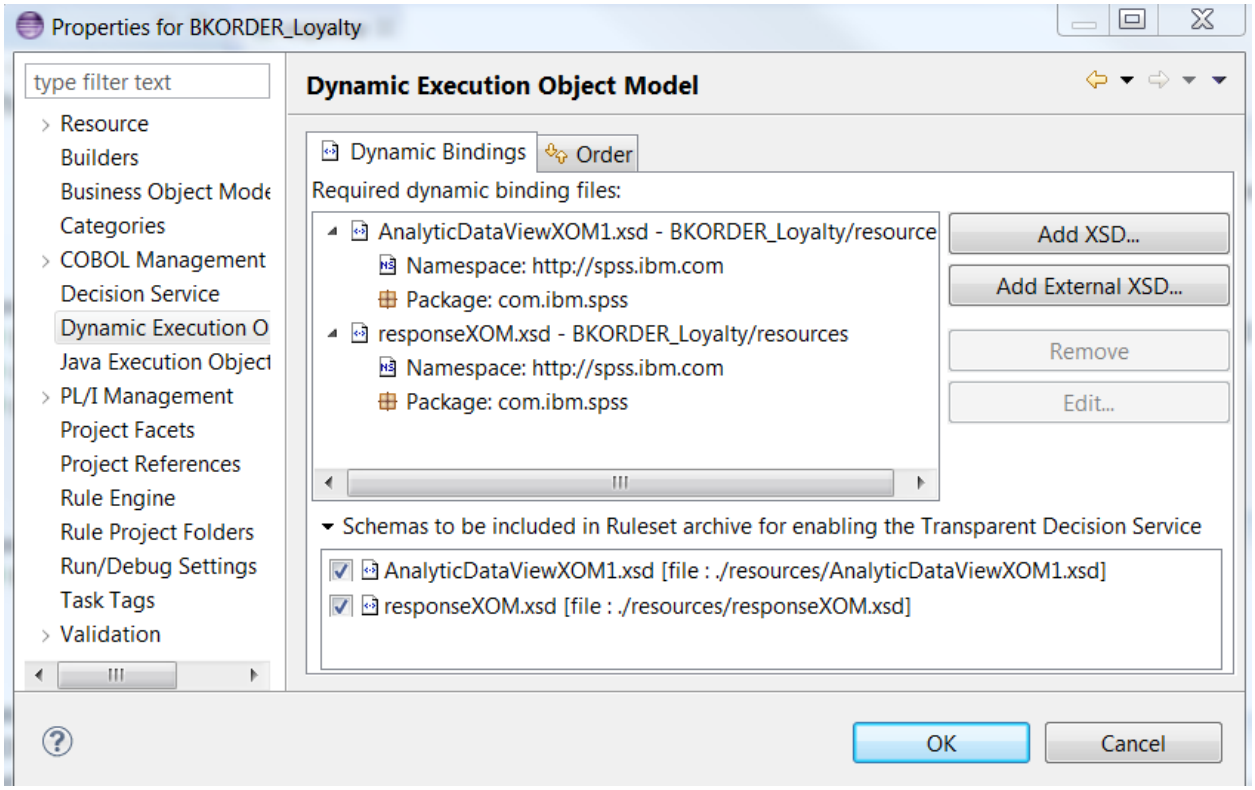
Figure 2.15 Rule Project after SPSS Artifact import



This figure shows the errors and also the schemas that have been imported from the ADV. SPSS provides a separate schema (responseXOM.xsd) that holds the properties that the rules will return. This response XOM template needs to be customized and updated before it can be used. For this scenario the LoyaltyDecision ruleset will return a "SuggestedLoyalty" string and a "TotalRevenue" value.

To resolve the BOM errors the schemas and XOMs need to be included in the project. The XOMs can be created by adding the schemas through the project map or project properties as shown in figure 2.16.

Figure 2.16 Importing the SPSS XOM into a Rule Project



After the XOMs have been registered most of the errors in the project should disappear.

Note also that enumerations and domains exported from SPSS (such as the GENDER and ORDERPLACED classes above) are not compatible with the ODM BOM and will need to be deleted and an alternate implementation for the domains established.

To create the domain for the Gender attribute follow these steps:

1. open the attribute in the BOM editor, and
2. click **Create a domain**
3. select “Literals” as type of domain
4. add ‘M’ as a literal
5. add ‘F’ as a literal
6. Navigate to the root package for the BOM: com.ibm.spss
7. Select the GENDER class and click Delete
8. Save the BOM.

To create the domain for the orderPlaced attribute follow these steps:

1. open the attribute in the BOM editor, and
2. click **Create a domain**
3. select “Literals” as type of domain
4. add ‘Y’ as a literal

5. add ‘N’ as a literal
6. Navigate to the root package for the BOM: com.ibm.spss
7. Select the ORDERPLACED class and click Delete
8. Save the BOM.

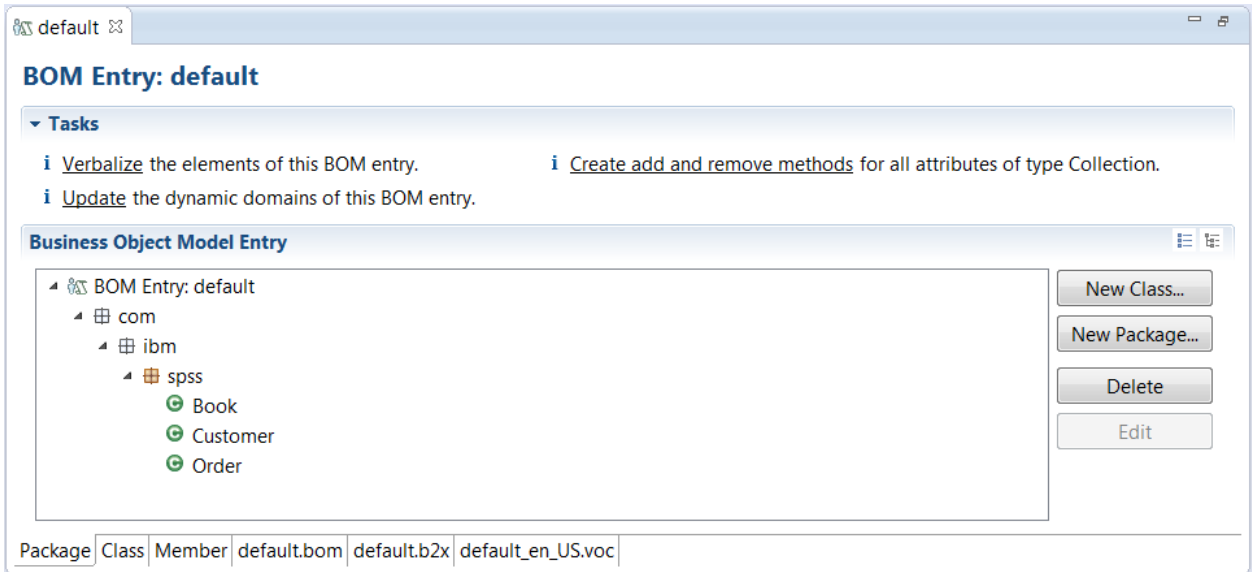
This should result in a **gender** attribute as shown below.

Figure 2.17 Establishing domain values for attributes

This approach will constrain the value of the **gender** attribute to being M or F in any rules. A similar approach is used to constrain the **orderPlaced** attribute to be Y or N.

The classes used to input information into the decision are now defined in the default BOM entry as shown below.

Figure 2.18 Default BOM entry after refactoring



The next step requires the response XOM properties to be added to the BOM. SPSS exports a template for the response in the file responseXOM.xsd and expects each attribute returned to be defined as an element in this schema. In this case we wish to return a "SuggestedLoyalty" attribute which is a string with values NONE, GOLD, SILVER, BRONZE and a "TotalRevenue" attribute which is calculated on the basis of past orders.

To establish this:

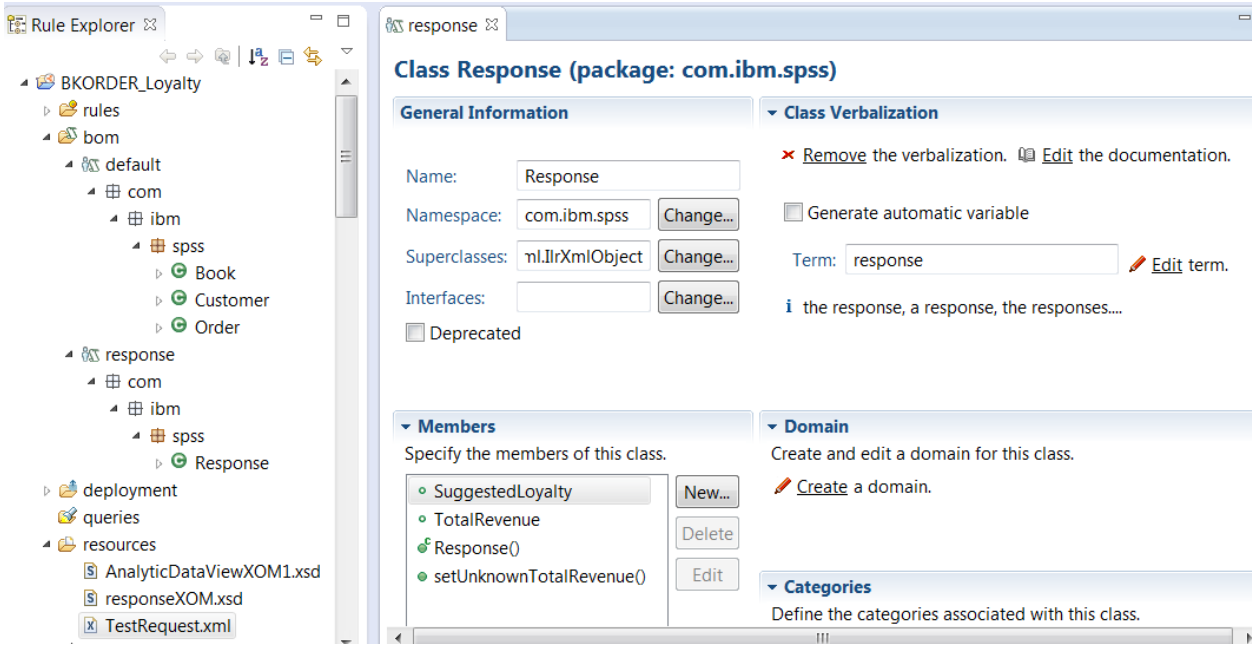
1. Change the "value" placeholder element in the response type to "SuggestedLoyalty" of type xsd:string.
2. Add a second element titled TotalRevenue of type xsd:double.
3. Save the responseXOM.xsd
4. Select the BKORDER_Loyalty project and right click – New -> BOM Entry
5. In the BOM entry Name field type response
6. Leave "Create a BOM entry from a XOM" checked and click **Next**.
7. Click Browse XOM and select the responseXOM.xsd you just modified
8. Select the Response class
9. Click Finish to create the response BOM entry

To create the domain for the SuggestedLoyalty attribute follow these steps:

1. Open the attribute in the BOM editor,
2. click **Create a domain**
3. select "Literals" as type of domain
4. add 'NONE' as a literal
5. add 'BRONZE' as a literal
6. add 'SILVER' as a literal
7. add 'GOLD' as a literal
8. Save the BOM.

This now completes the BKORDER_Loyalty BOM as shown in figure 2.20.

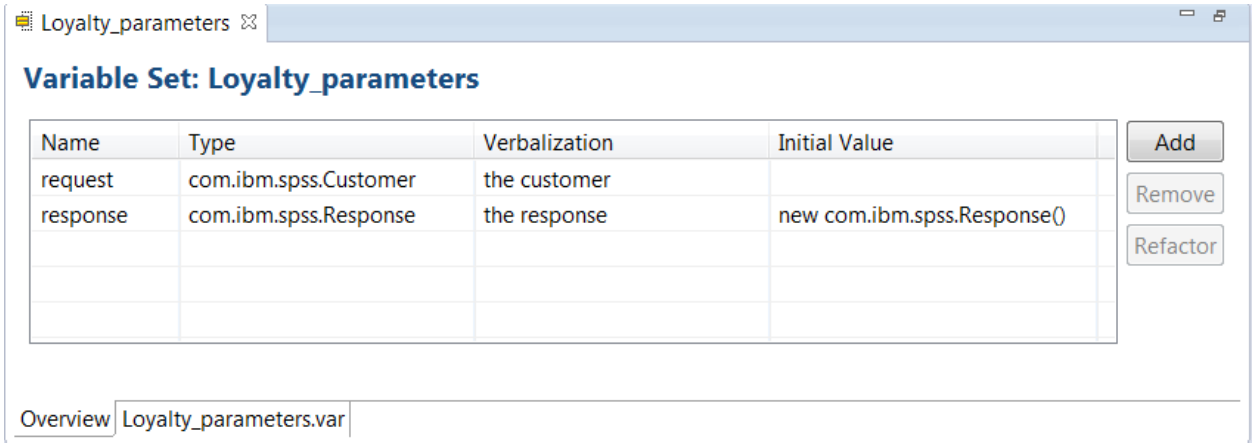
Figure 2.20 Completed BKORDER_Loyalty BOM



Creating the BKORDER_LOYALTY Ruleset

Having agreed the Object Model to be used in the rules that rank customers loyalty, the next step is to setup the signature of the ruleset defining the input and output parameters to be used. The variable set that defines the input and output terms are first defined.

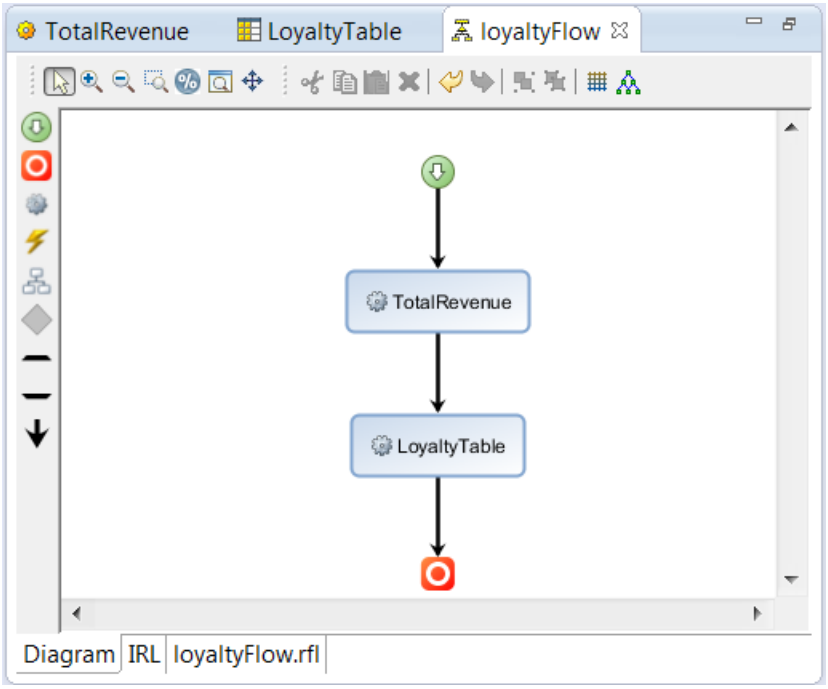
Figure 2.21 Loyalty_parameters variable set



In this case the request is in the form of a customer with an order history and the response is simply the Response element that was defined in the BOM with the **SuggestedLoyalty** and **totalRevenue** attributes. As the response is not supplied by SPSS it needs to be initialized before being referenced otherwise a null pointer exception will result when the rules reference the variable.

With these parameters we can then write some rules that will classify the customer based on the analytics information. These are very simple in the scenario and consist of an action rule that calculates the revenue for each customer, followed by a LoyaltyTable Decision table that allocates a suggested loyalty based on their total revenue. These rules are orchestrated in a loyaltyFlow ruleflow as shown below.

Figure 2.22 Loyalty Flow ruleflow



The ruleflow invokes two artifacts:

- TotalRevenue sums the total orders for the customer as shown in figure 2.23.
- LoyaltyTable provides a recommendation for the loyalty of the customer as shown in figure 2.24.

Figure 2.23 Total Revenue Action Rule

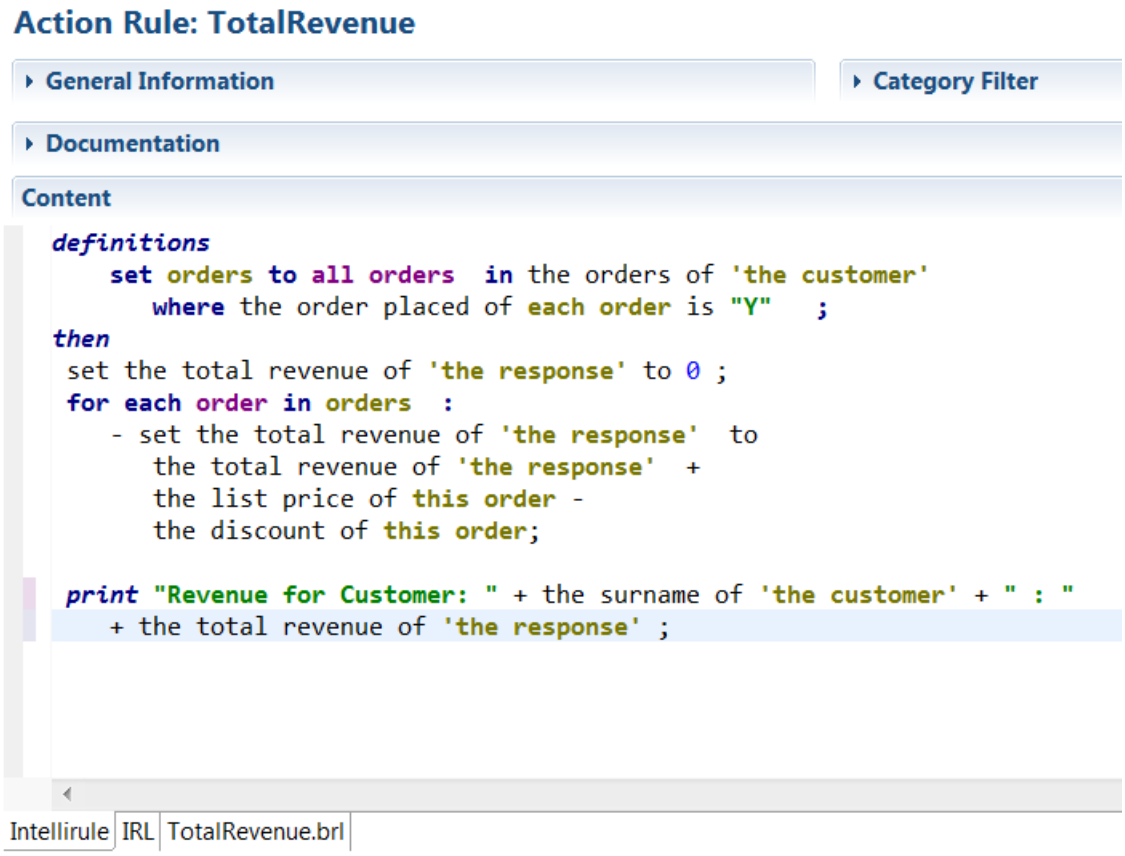
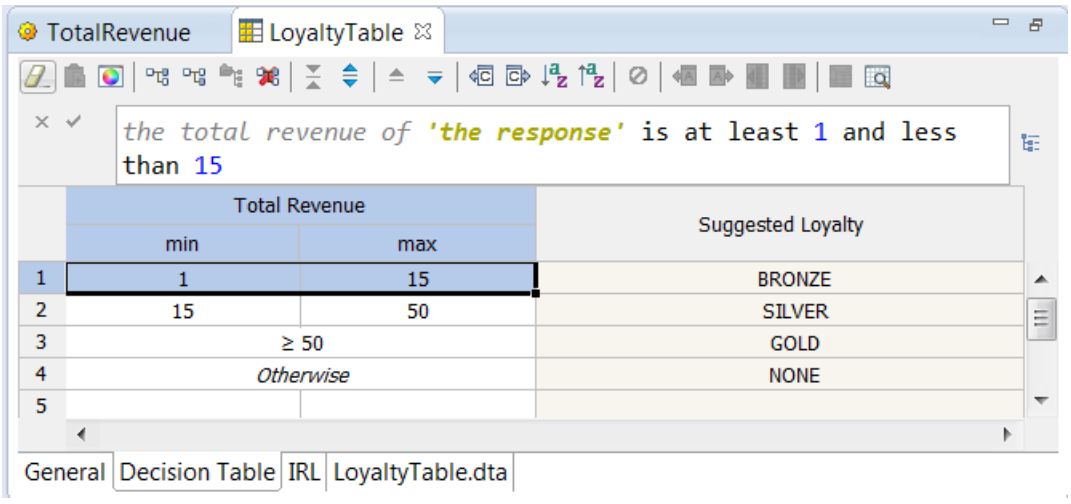
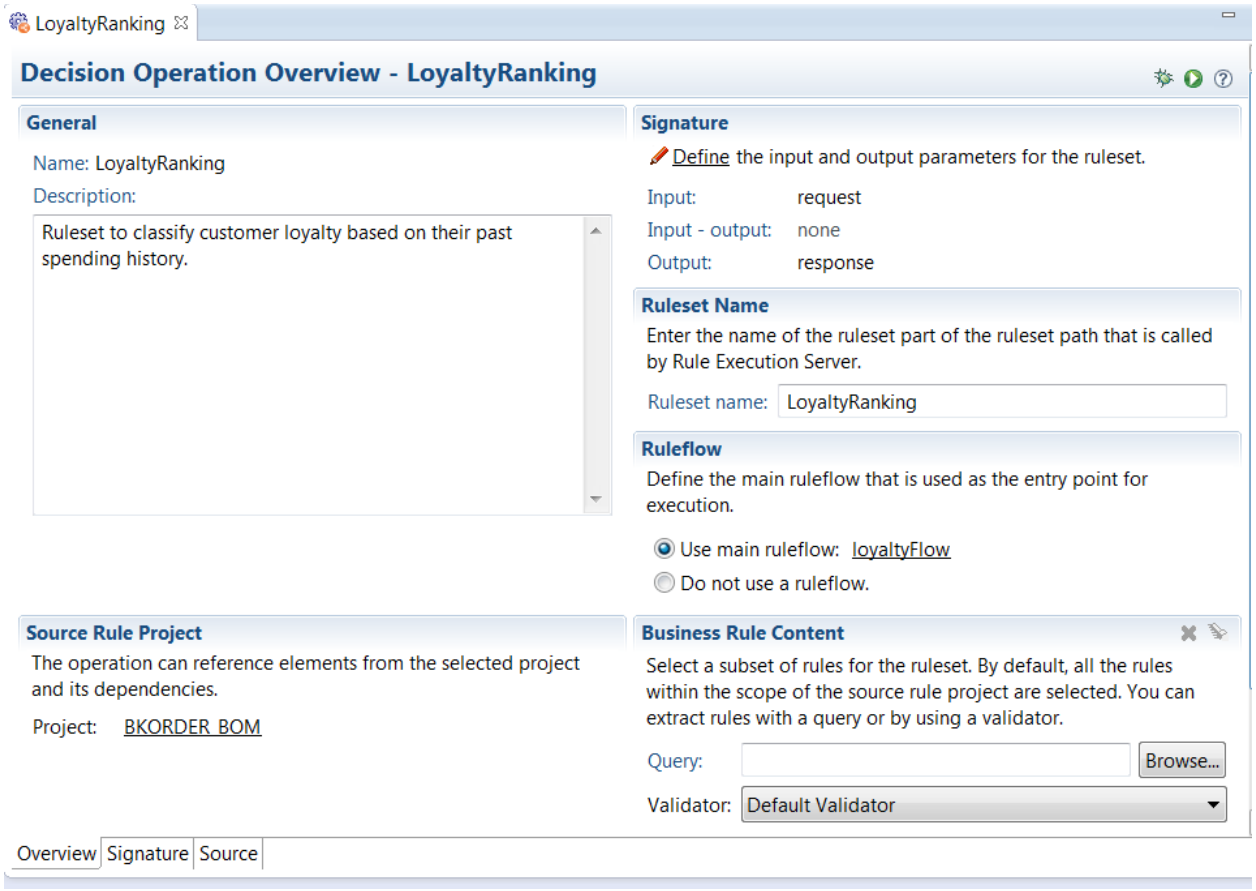


Figure 2.24. Loyalty Table Decision Table



To define the ruleset within the decision service, we need to take these rules and package them using a decision operation as shown in figure 2.25.

Figure 2.25 Loyalty Ranking decision operation



In this case you define the input parameter to be the Customer variable based on the ADV. This will contain all the customer and order records. The response output parameter is based on the Response class. This is initialized in the variable set (as shown in figure 2.21) so that it can be modified by the rules without a null pointer exception.

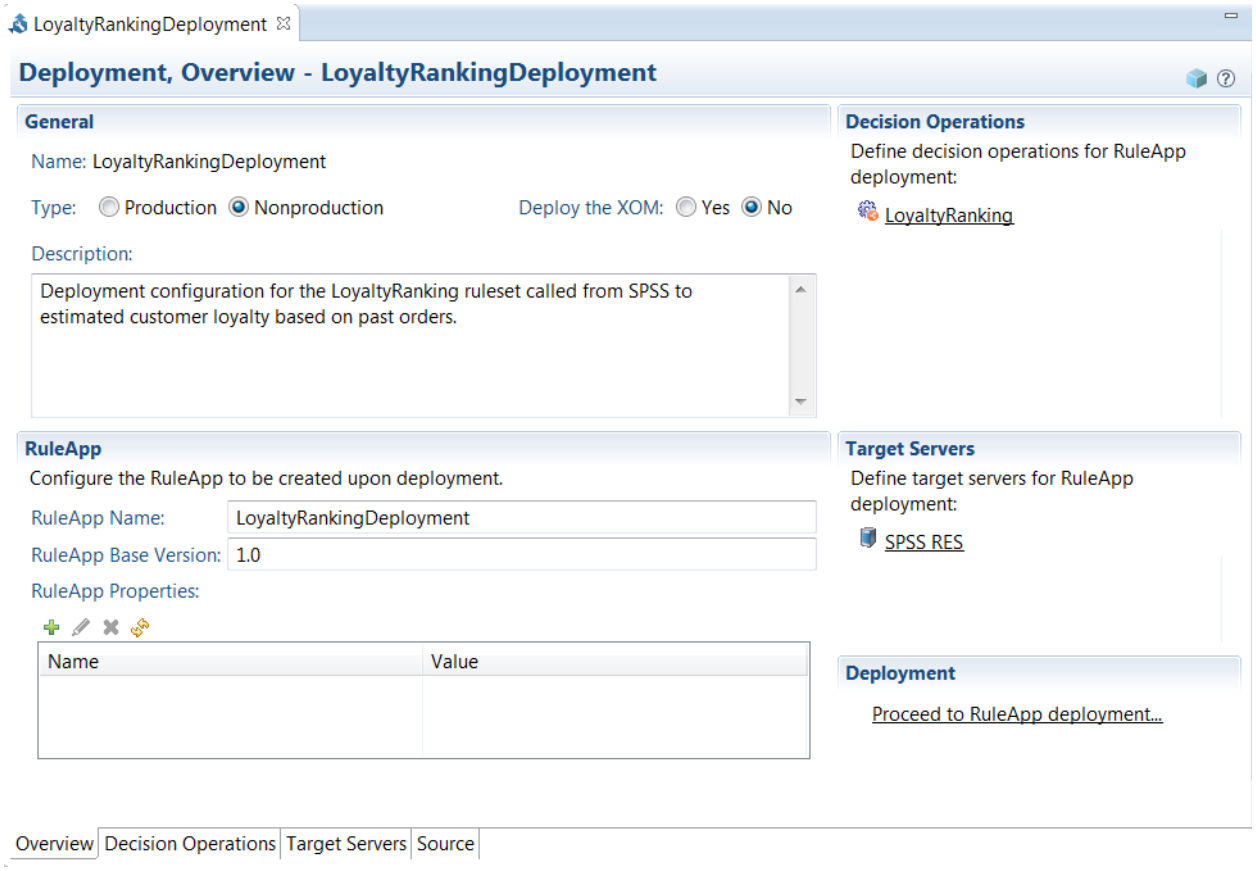
Deploying the LoyaltyRanking Ruleset

Once the LoyaltyRanking ruleset has been developed, it must be deployed to the decision server that is integrated with SPSS. In this scenario we will include this ruleset or operation into the main BookOrderDecisionService that already contains the BookOrderPricing decision.

In this reference environment the C&DS AppSrv01 profile has been augmented with a Decision Server Rules profile template allowing the Rule Execution Server and thus the rulesets, to run on the same application server as SPSS. Once the ruleset is deployed it is visible in the Rule Execution Server console.

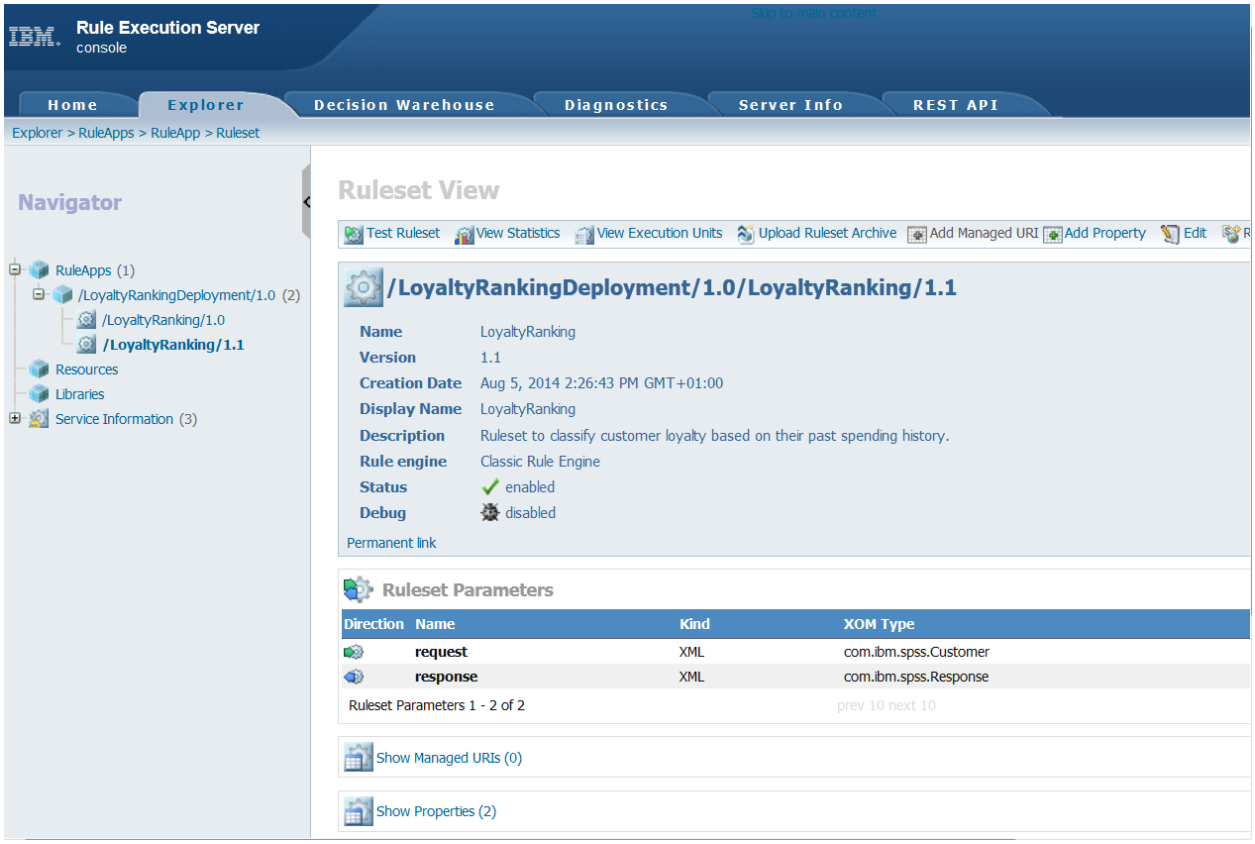
To enable the deployment, a second deployment configuration is created that specifies the LoyaltyRanking operation (ruleset) to be deployed and the SPSS RES that is the target decision server it is to be deployed to.

Figure 2.26 LoyaltyRankingDeployment deployment configuration.



On clicking **Proceed to RuleApp deployment**, the ruleset will be deployed to the Rule Execution Server that is integrated with SPSS.

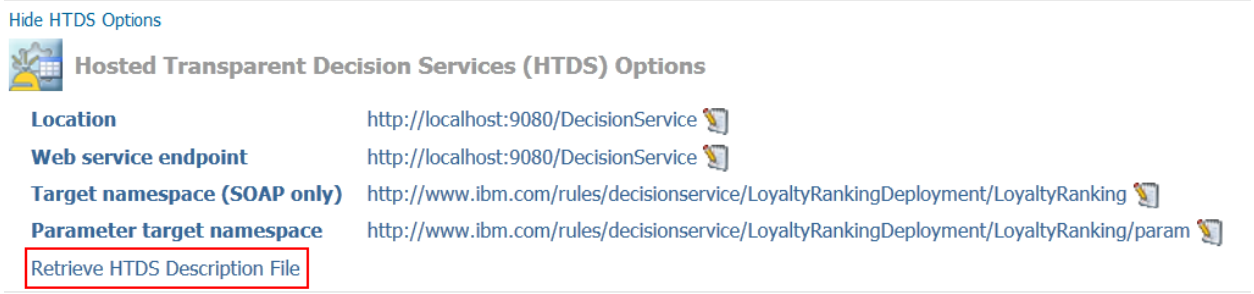
Figure 2.27 LoyaltyRanking ruleset deployed to SPSS Decision Server

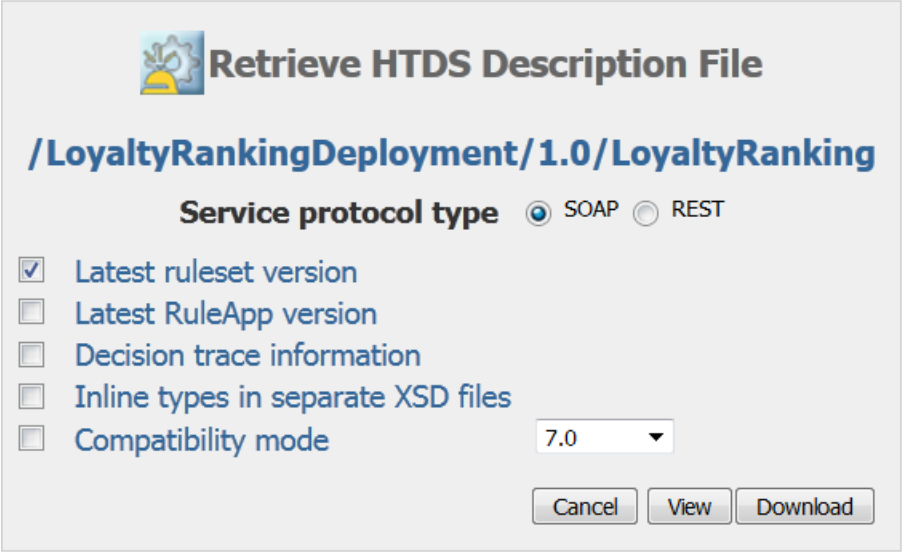


Once the ruleset is deployed you should note the ruleset path used to identify the ruleset. This determines the endpoint used to invoke the ruleset from SPSS. For example: /LoyaltyRankingDeployment/1.0/LoyaltyRanking/1.1

In order to integrate with SPSS you need to obtain the WSDL that describes the HTDS web service. In order to obtain the WSDL, you should select the **Show HTDS options** (as highlighted below in figure 2.28) to reveal the link **Retrieve HTDS Description file**.

Figure 2.28 Retrieve HTDS Description File



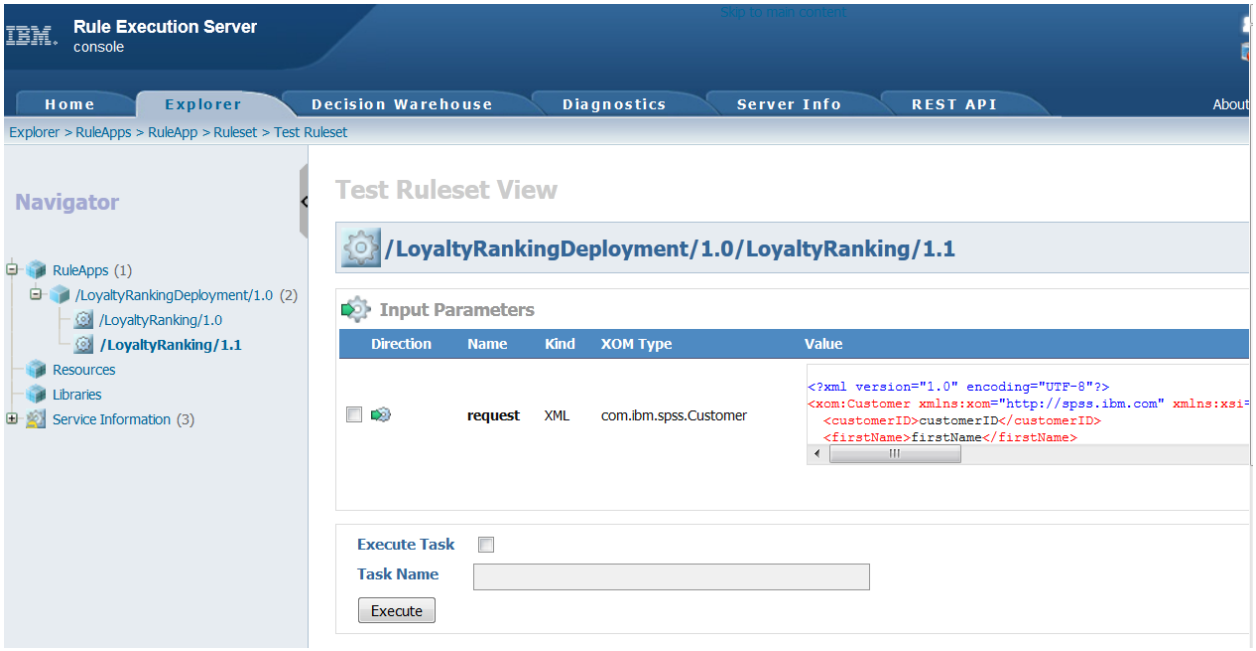


In the pop up ensure SOAP is selected and latest ruleset version. This then means that the latest rulesets are used when redeployed from Rule Designer or Decision Center (assuming on deployment the ruleset version is incremented).
Click the **View** button to view and obtain the url to download the WSDL for this ruleset.

The url needed to access the web service can be obtained from the browser window and in this case is:
http://localhost:9080/DecisionService/ws/LoyaltyRankingDeployment/1.0/LoyaltyRanking?WSDL

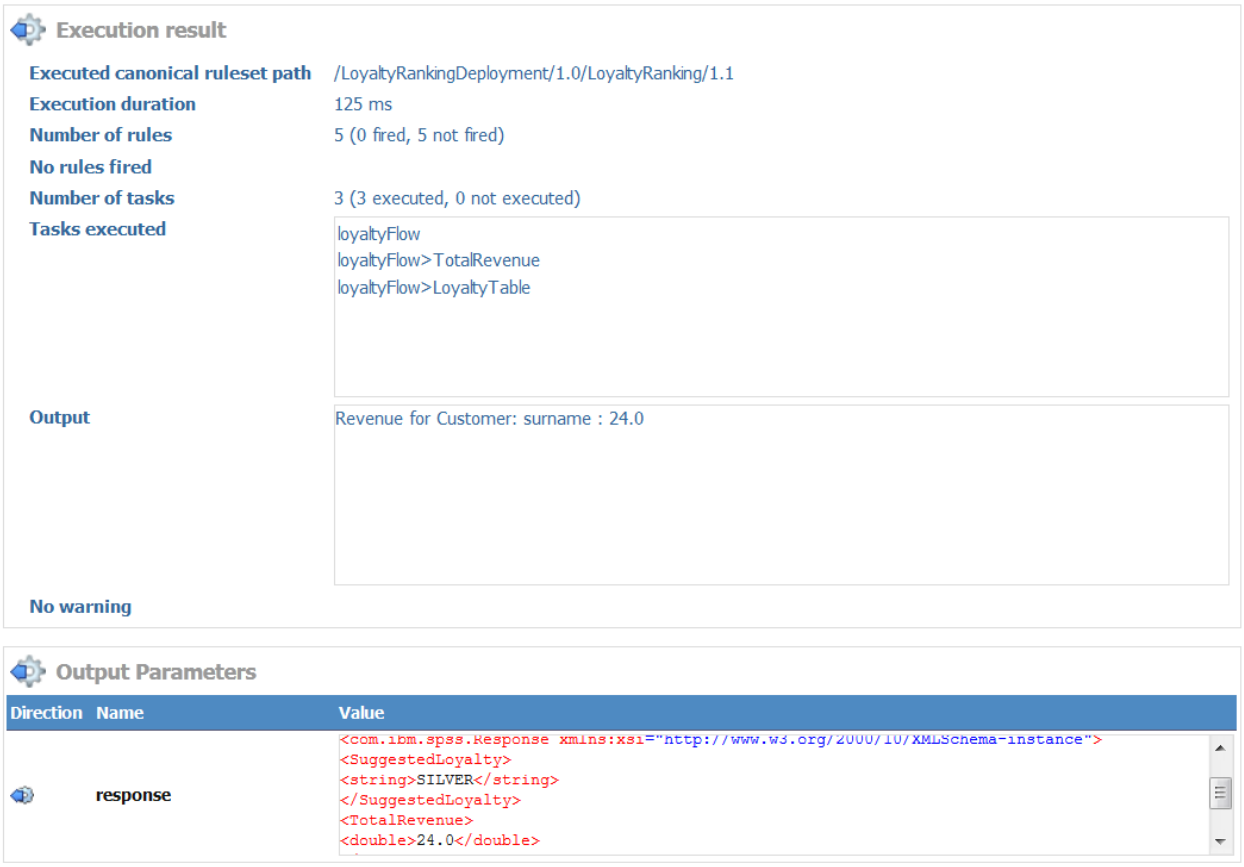
The ruleset should be tested using the Rule Execution Server console to ensure that it functions correctly. An XML customer request can be generated from the request schema using eclipse tooling and can be pasted into the request panel in the Rule Execution Server console.

Figure 2.29 Testing a ruleset in the RES Console



On clicking Execute the ruleset results are displayed.

Figure 2.30 Ruleset Execution results

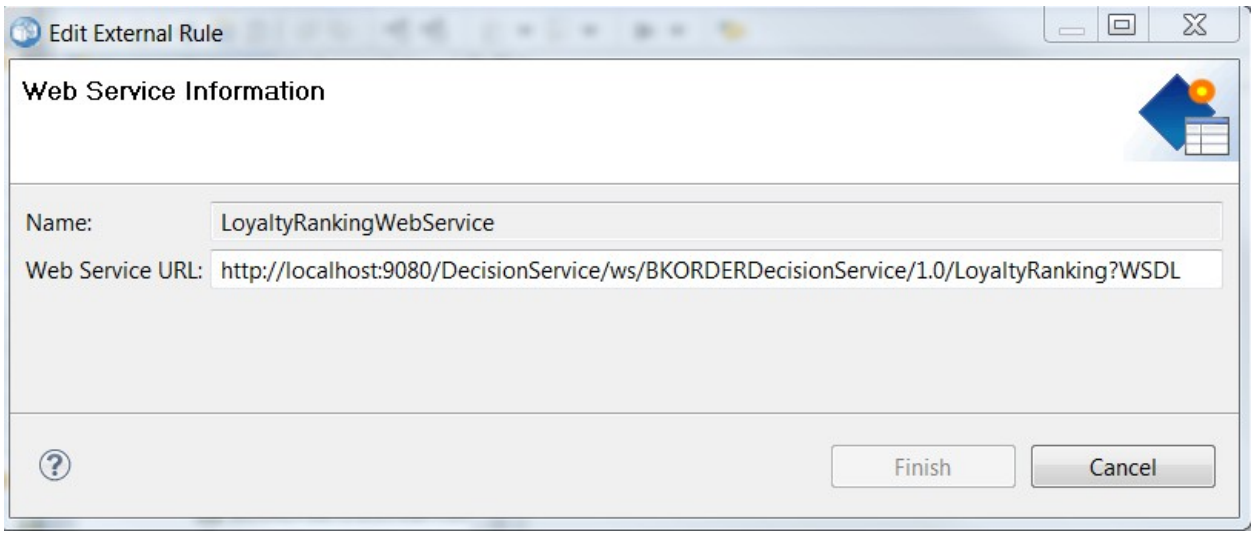


This ODM HTDS decision service can now be imported into SPSS as an **External Rule** and used as part of an SPSS Modeler stream. To create an external rule you need to use the C&DS Deployment Manager using the following steps:

1. Open C&DS Deployment manager
2. Navigate to your project folder in the Content Repository e.g. **Content Repository > BookOrder**
3. Right click and select New > External Rule
4. In the external Rule information ensure Web Service is selected
5. Enter a Name for the Rule LoyaltyRankingWebService
6. Click Next
7. In the Web Service URL enter the url for the HTDS web service: http://localhost:9080/DecisionService/ws/LoyaltyRankingDeployment/1.0/LoyaltyRanking?WSDL
8. Click **Finish**

At this point C&DS will import the WSDL from the Rule Execution Server and identify the input output parameters and web service endpoint. This can be updated by editing the External Rule as shown in figure 2.31.

Figure 2.31 Editing a Web Service External Rule



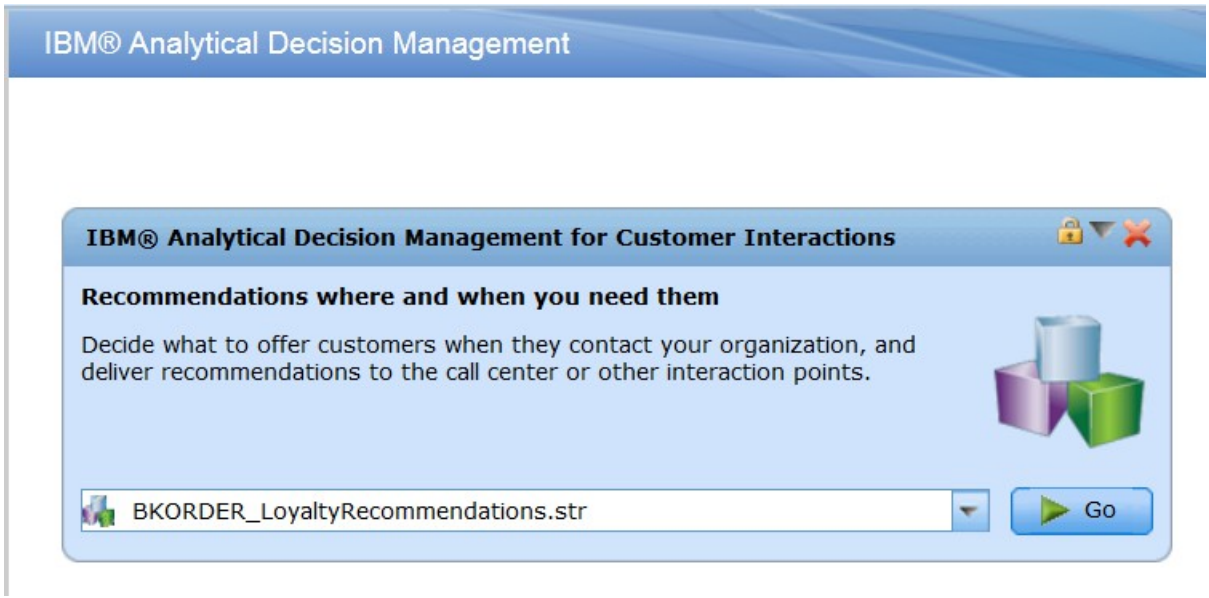
This web service can now be referenced in Analytical Decision Management applications.

Configuring an Analytical Decision Management Project

Before we can show how SPSS can invoke ODM as part of the Analytic Decision Management applications we have to create an application project. For our scenario we will use the IBM Analytical Decision Management for Customer Interactions application.

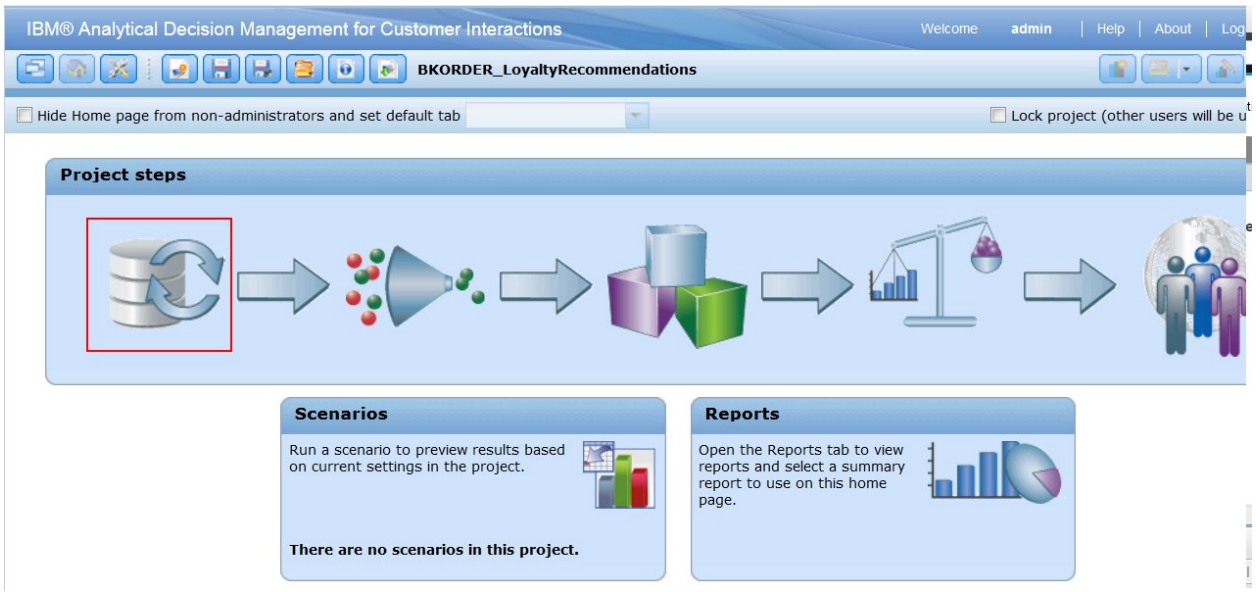
Open IBM Analytical Decision Management (<host>:<port>/DM) and create a new application (BKORDER_LoyaltyRecommendations) as shown below:

Figure 2.32. Analytical Decision Management Home Page



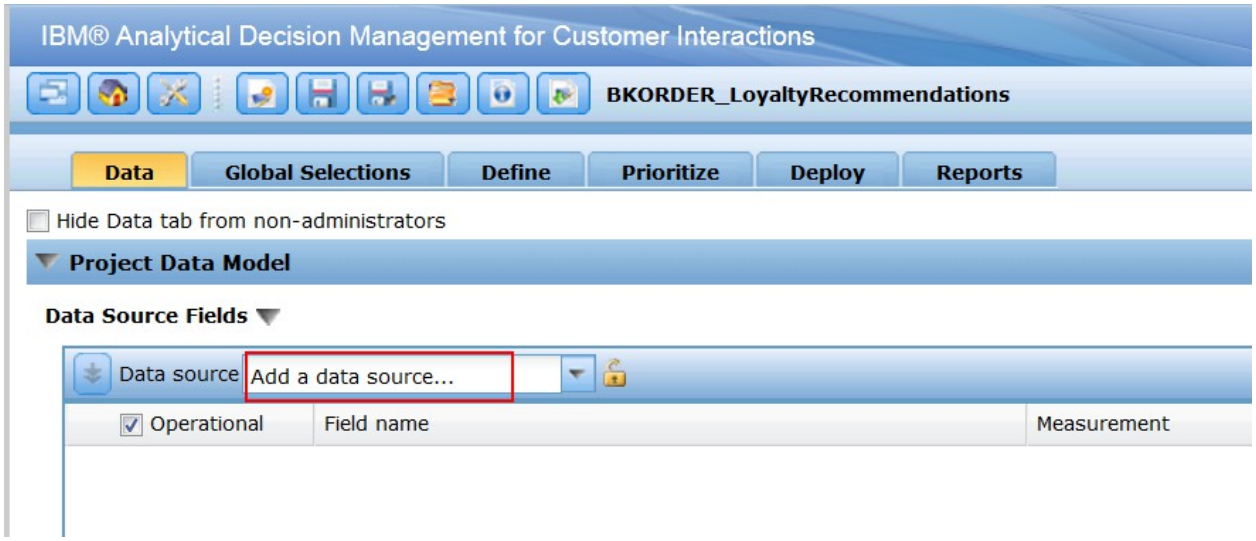
Once the new application is named and saved, open it and select the Data step. From here you can define the data used in the application from the BKORDER_ADV created previously.

Figure 2.33. BKORDER_LoyaltyRecommendations Project



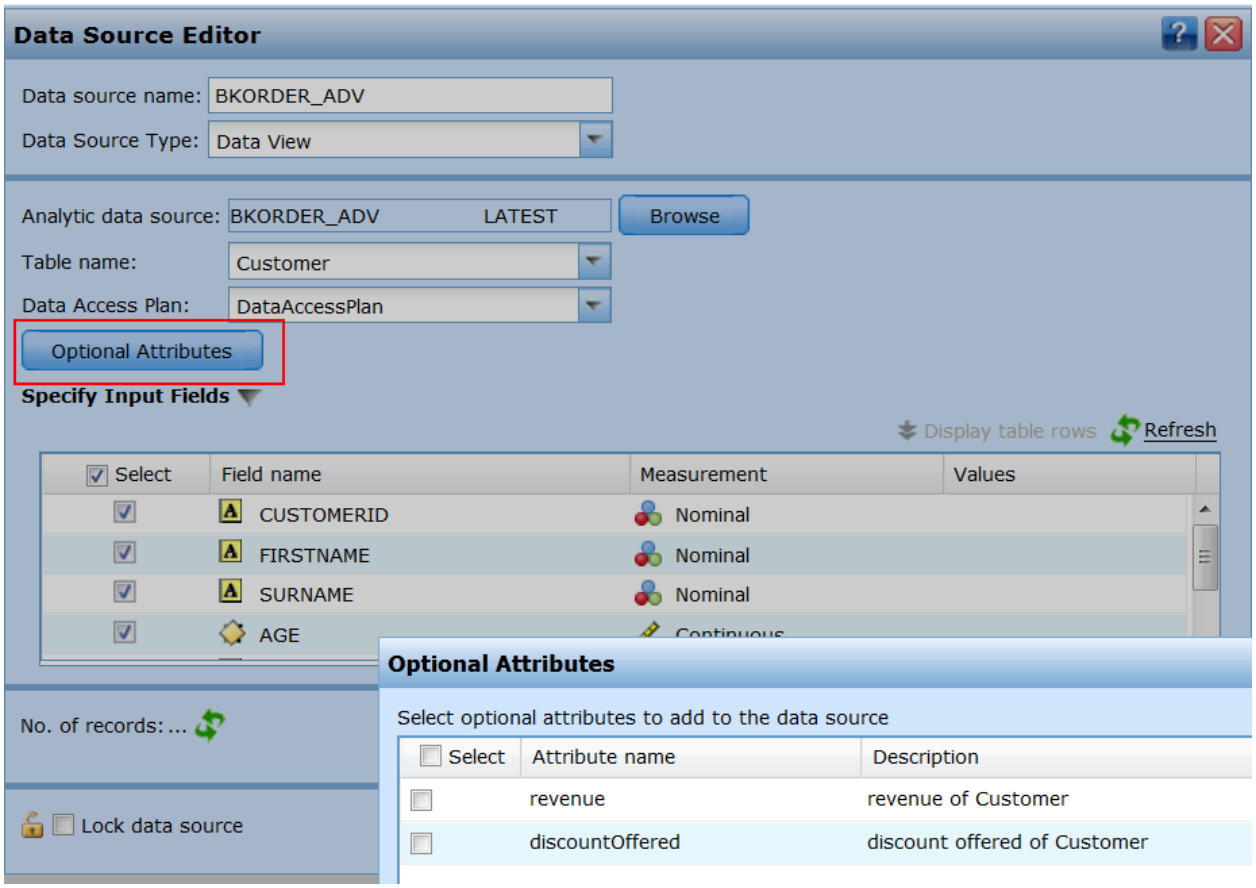
In the Data tab > Project Data Model > Data Source pull down select **Add a data source** …..

Figure 2.34. Adding a Data Source to an ADM project



You can then configure the ADV in the Data Source Editor.

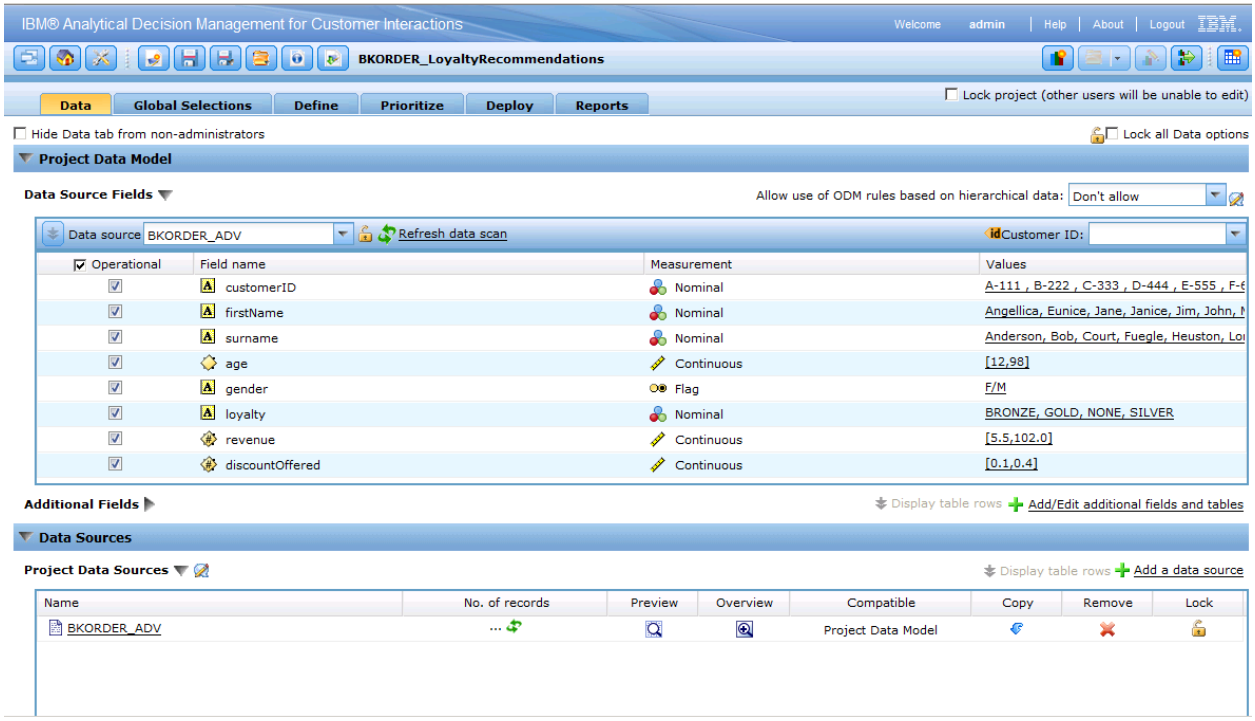
Figure 2.35. ADM Project Data Model Data Source Editor



The configuration options are identical to those used for the ADV stream in the previous section. You can also set-up the optional derived attributes so these can be used in deciding what loyalty cards to offer to customers.

On saving the data source and refreshing the data scan, a list of values for each field in the ADV is shown.

Figure 2.36. Project Data Model Data Source

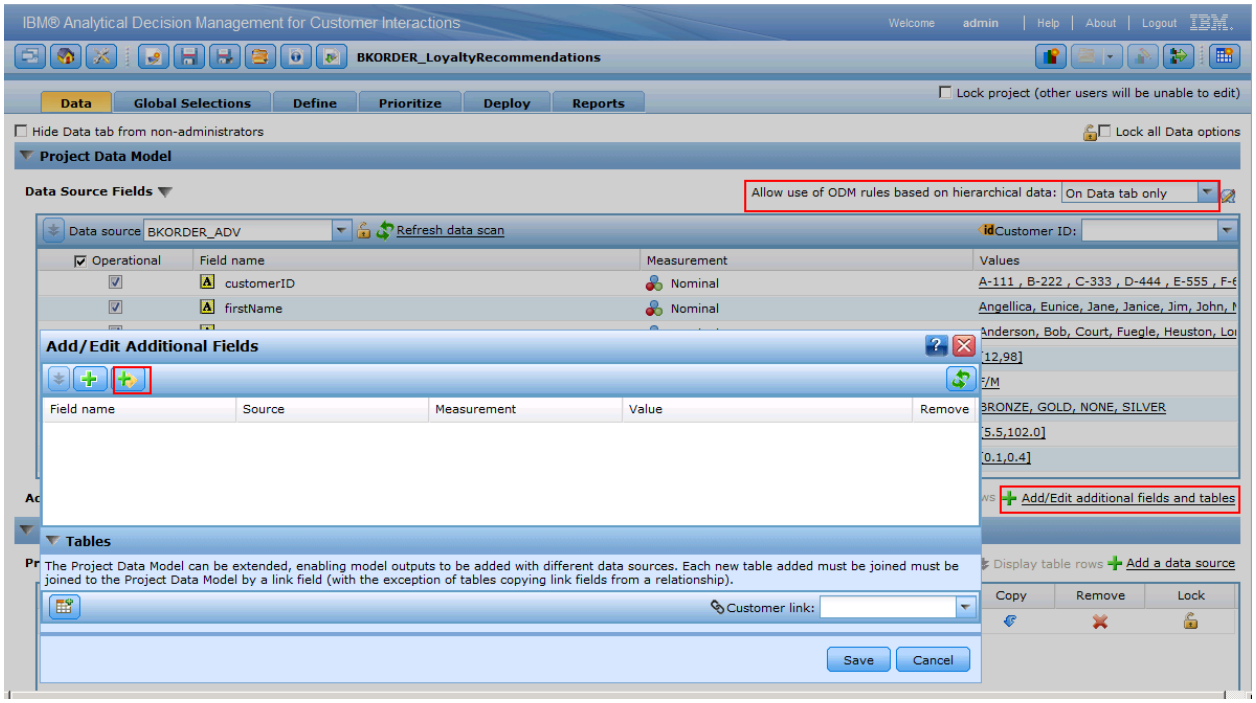


At this point we have the BKORDER_ADV in the project and can use it to provide the information to ODM on which to make the decision. The response from the rules can then be merged into the data available to the project by allowing use of ODM rules and adding additional fields returned from the decision service call.

Integrating an External Rule/ decision service call into the ADM Project Data model

To integrate ODM rules into an application we use the Project Data Model panel in the project Data tab.

Figure 2.37 Adding the decision service response to a project



To add the response from the ODM decision service to the data perform the following steps

1. Select “On Data tab only” for the **Allow use of ODM rules based on hierarchical data pull-down**
2. Click **Add/Edit additional fields and tables**
3. In the pop-up box that results click the add model output icon.

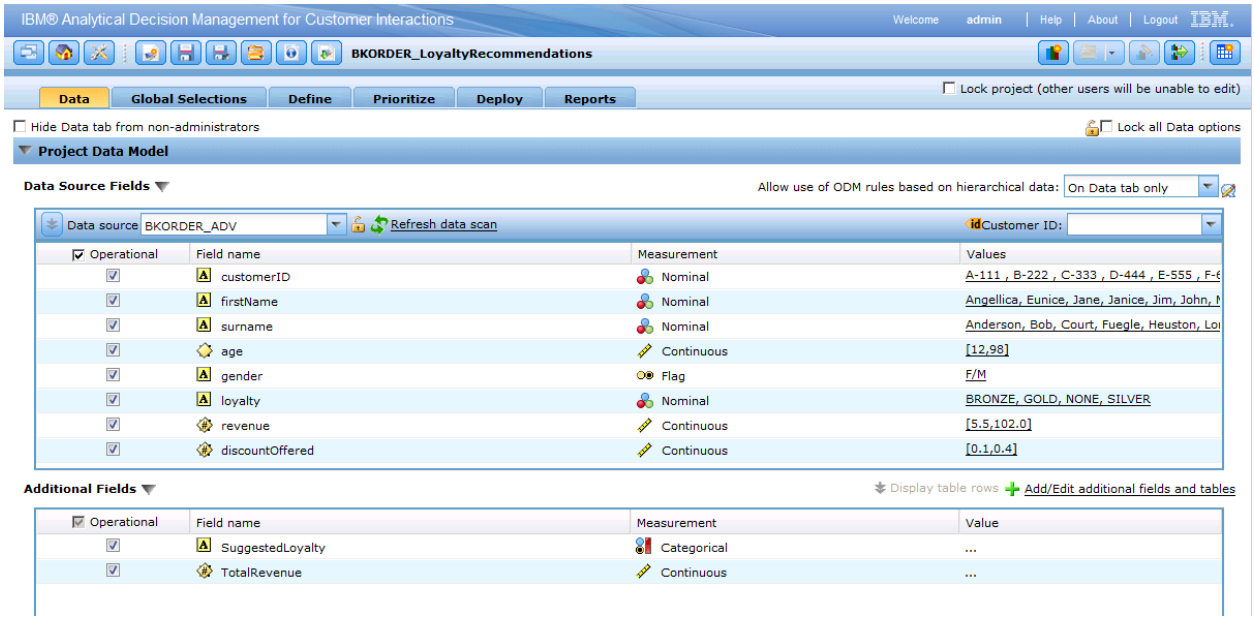
This will present a browser that allows you to include data from other models including the External Rules that you have just created.

To add the output parameters follow these steps:

1. In the **Find a model** pop-up browse to the Content Repository/BookOrder folder and select the *LoyaltyRankingWebService*
2. Click **Open**
3. Select the two fields SuggestedLoyalty and TotalRevenue
4. Click Save
5. Click Save in the Add/Edit Additional Fields

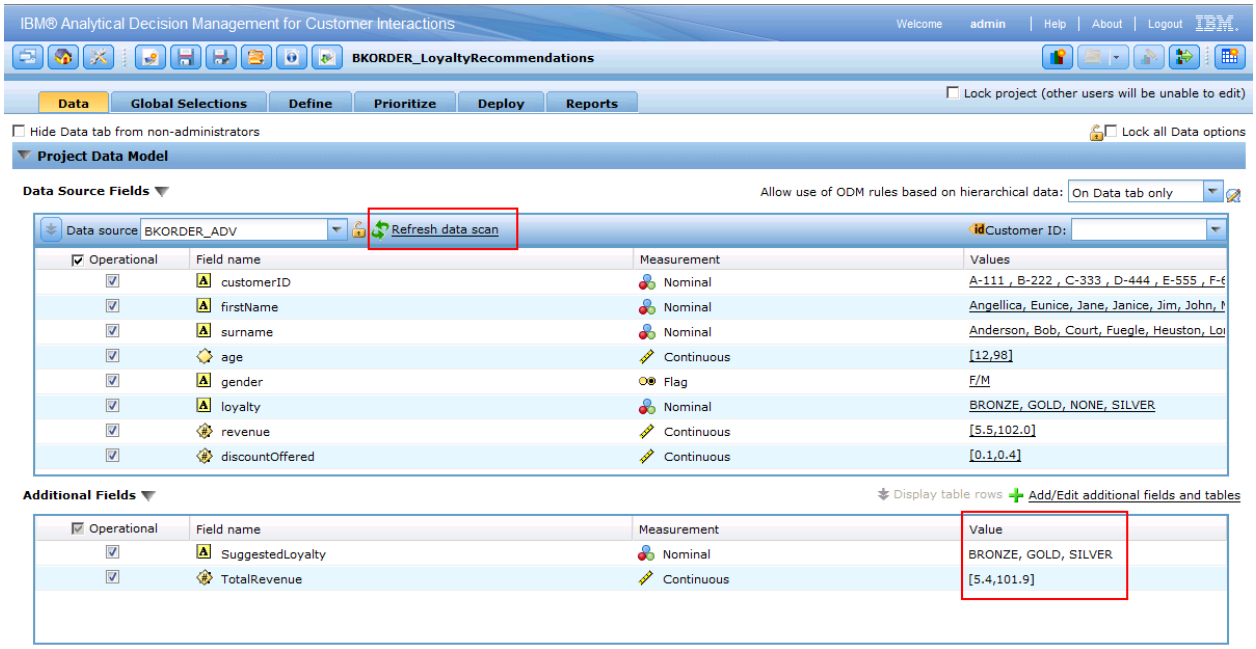
This now provides additional fields that can be used in the Project Data Model.

Figure 2.38 ADM Project Data View with External Rule additional fields



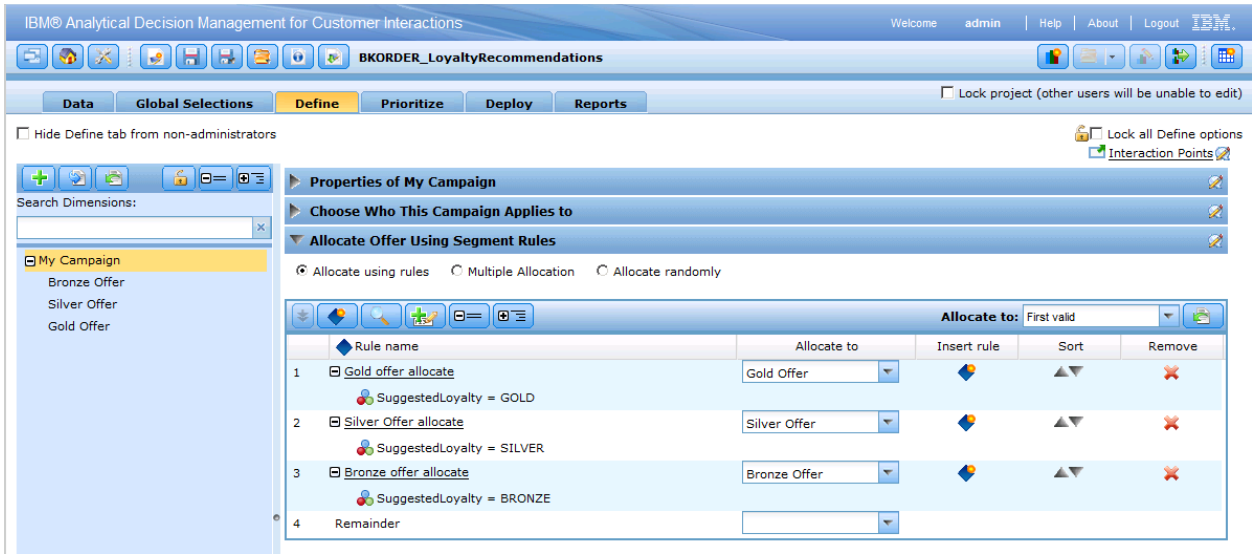
With this configuration when **Refresh data scan** is clicked, the BookCustomerADV stream is passed to the ODM decision service and for each customer a loyalty and total order value are calculated.

Figure 2.39 Project Data Model Refresh data scan using External Rules



The SuggestedLoyalty field can be used to allocate customers to certain offers in the application. These offers should be set-up in the Define tab as shown below.

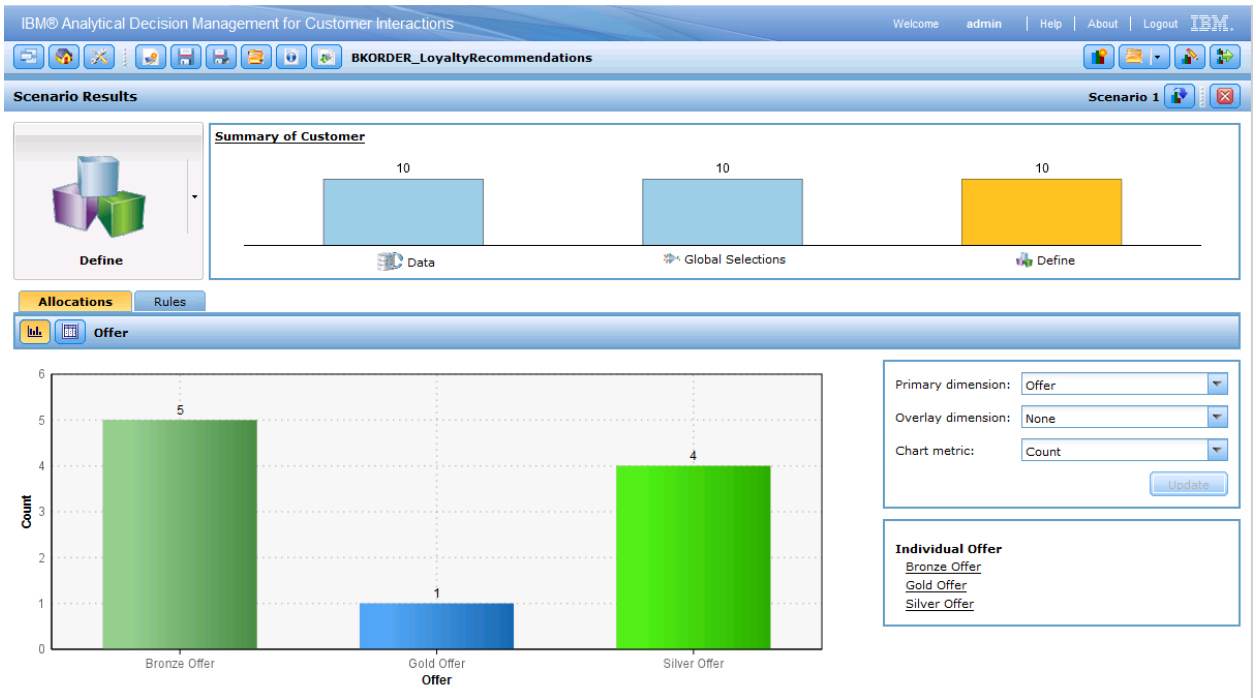
Figure 2.40 Setting up the ADM Campaign to use the response of ODM



In this case there is a rule for each offer that allocates the offer on the basis of its respective SuggestedLoyalty flag.

When this campaign is run in a scenario the allocation of the customers to an Offer can be quickly seen.

Figure 2.41 Offer Allocation based on Loyalty rules from ODM



This shows a fairly simple distribution based only on the SuggestedLoyalty provided by ODM. In a real solution there would be much more information available and more sophisticated rules used to calculate the loyalty.

Pattern 3 – Predictive Scoring Services

The goal of this pattern is to show how a predictive analytics model can be used to make operational decisions more effective. In the previous analytics pattern the customer loyalty was derived based on a combination of analytics and rules. This was then used by ADM for determining the criteria for offering loyalty cards. In situations where operational decisions are being made, it may be useful to know not only what loyalty the customer has, (as determined by the 360 degree view of the customer) but also how likely a customer is to buy based on their loyalty. This can then be used to influence the discounts or offers that are made in the pricing decision service at runtime.

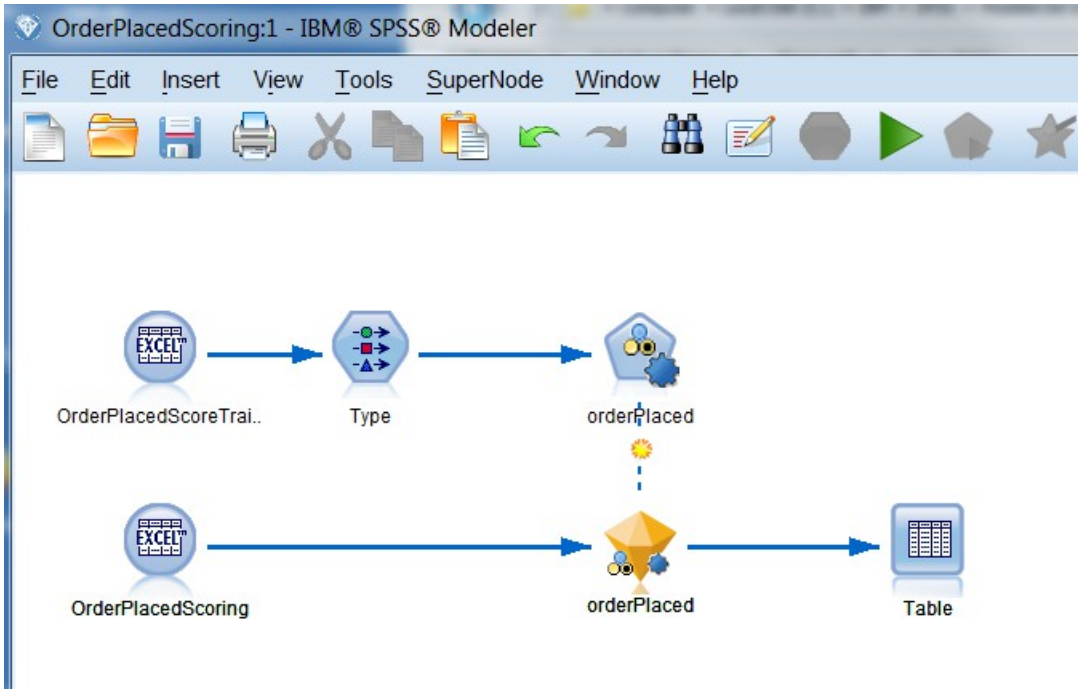
This section provides a simple overview of how to develop and use a scoring model for those readers with little analytics experience. This example uses the base facilities of SPSS Modeler to explain how the model is produced and thus the implications of its use. In practice the Modeler Advantage product may be an easier approach to developing a scoring model.

Predictive Analytics Model

SPSS processes information and applies analytic functions using streams. These streams take information from a variety of sources and produce outputs that can be used in other streams or decision making applications.

For the predictive scoring pattern we provide an Order Placed Scoring Stream which is used to produce a model that predicts whether a customer will place an order based on their gender and loyalty.

Figure 3.1 Order Placed Scoring Stream

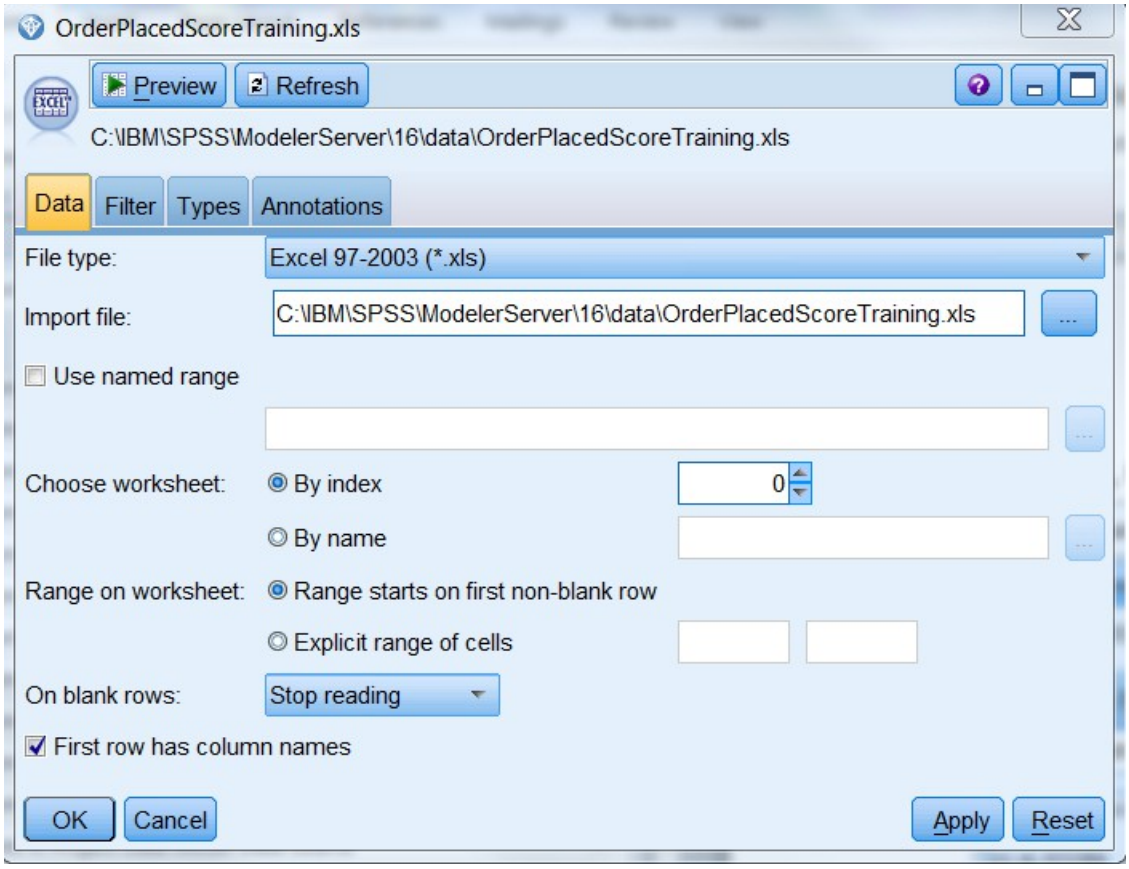


The stream provides the following nodes:

- **OrderPlacedScoreTraining** – this source provides training data with examples of whether an order was placed against customer gender and age. This information would normally come from a customer database and could include a much wider range of 360 degree information discovered through analytics.
- **Type** – this node indicates which fields in the source should be used as inputs to the model (gender and loyalty) and which should be the target for the prediction (orderPlaced).
- **orderPlaced** (Auto Classifier) – this model node is trained by the information from the source and produces a range of different models. It then selects the three that provide the best fit with the training data to produce the Loyalty Model or Nugget.
- **orderPlaced** (Nugget) – this model is produced by the auto classifier and represents the best model for predicting whether an order is placed when provided with customer loyalty and gender. This is the node that is executed in a scoring service.
- **OrderPlacedScoring** source defines the input signature for the orderPlaced (Nugget) and hence the scoring service. The orderPlaced field is filtered from the input data and is provided by the model / scoring service.

Each of these nodes is described in more detail to show how they are configured to produce the scoring model.

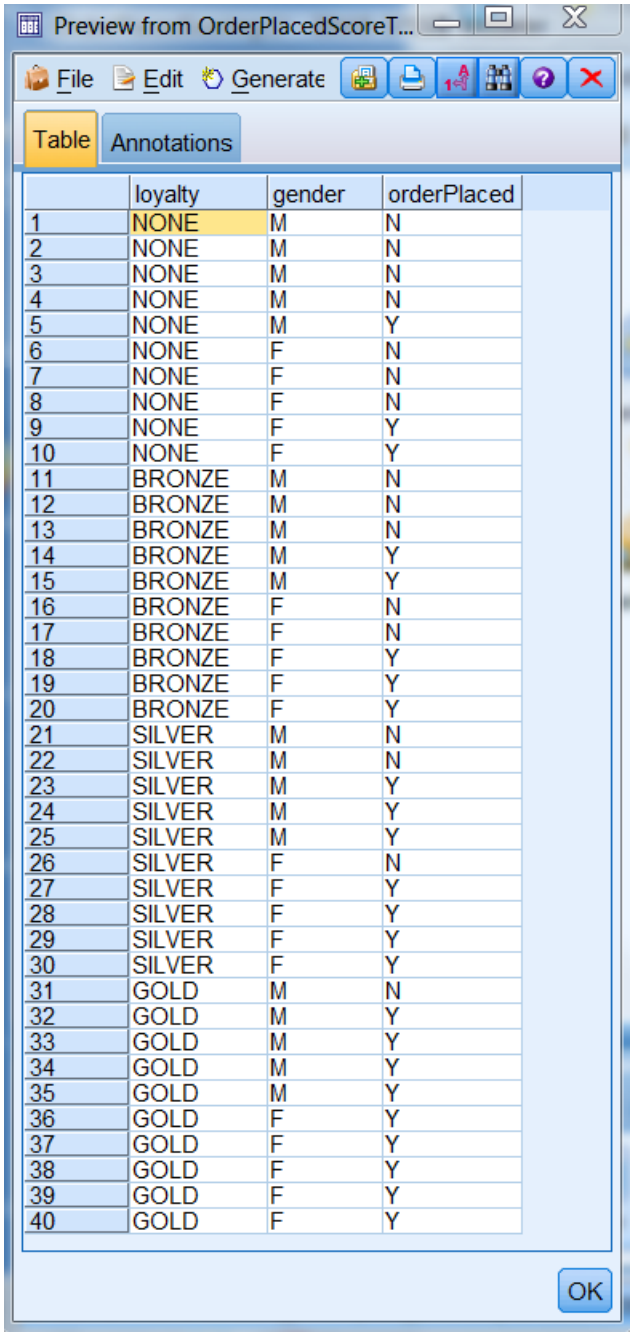
Figure 3.2 OrderPlacedScoreTraining Source



The OrderPlacedScoreTraining source is provided through an excel spreadsheet. For this scenario it allows the training data to be separated out and carefully controlled simplifying the overall solution. In reality this information should come from the customer 360 degree view where the loyalty of the customer is based on actual purchases.

The training data used here is very simple and can be shown in the File panel or by clicking the preview button.

Figure 3.3 OrderPlacedScoreTraining Source Data Preview

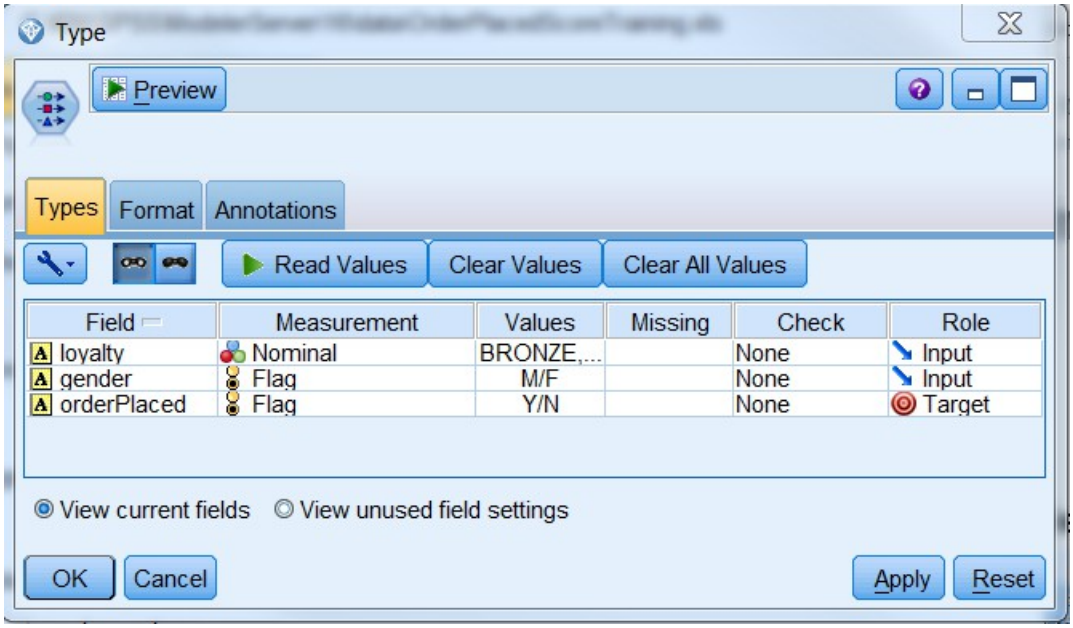


	loyalty	gender	orderPlaced
1	NONE	M	N
2	NONE	M	N
3	NONE	M	N
4	NONE	M	N
5	NONE	M	Y
6	NONE	F	N
7	NONE	F	N
8	NONE	F	N
9	NONE	F	Y
10	NONE	F	Y
11	BRONZE	M	N
12	BRONZE	M	N
13	BRONZE	M	N
14	BRONZE	M	Y
15	BRONZE	M	Y
16	BRONZE	F	N
17	BRONZE	F	N
18	BRONZE	F	Y
19	BRONZE	F	Y
20	BRONZE	F	Y
21	SILVER	M	N
22	SILVER	M	N
23	SILVER	M	Y
24	SILVER	M	Y
25	SILVER	M	Y
26	SILVER	F	N
27	SILVER	F	Y
28	SILVER	F	Y
29	SILVER	F	Y
30	SILVER	F	Y
31	GOLD	M	N
32	GOLD	M	Y
33	GOLD	M	Y
34	GOLD	M	Y
35	GOLD	M	Y
36	GOLD	F	Y
37	GOLD	F	Y
38	GOLD	F	Y
39	GOLD	F	Y
40	GOLD	F	Y

The preview views normally only show 10 records but this can be changed in the stream properties. The values in the scoring are not realistic or optimized but basically represent different distributions with females tending to buy more often than men and with a trend of GOLD customers being more likely to buy than lower ranked customers.

With so few records, the model cannot be trained well and this highlights the skill and care needed when developing predictive models.

Figure 3.4 Type Node and field role definitions



The Type node is used to work out the way the information from the source should be interpreted. This includes working out the measurement and type of each field:

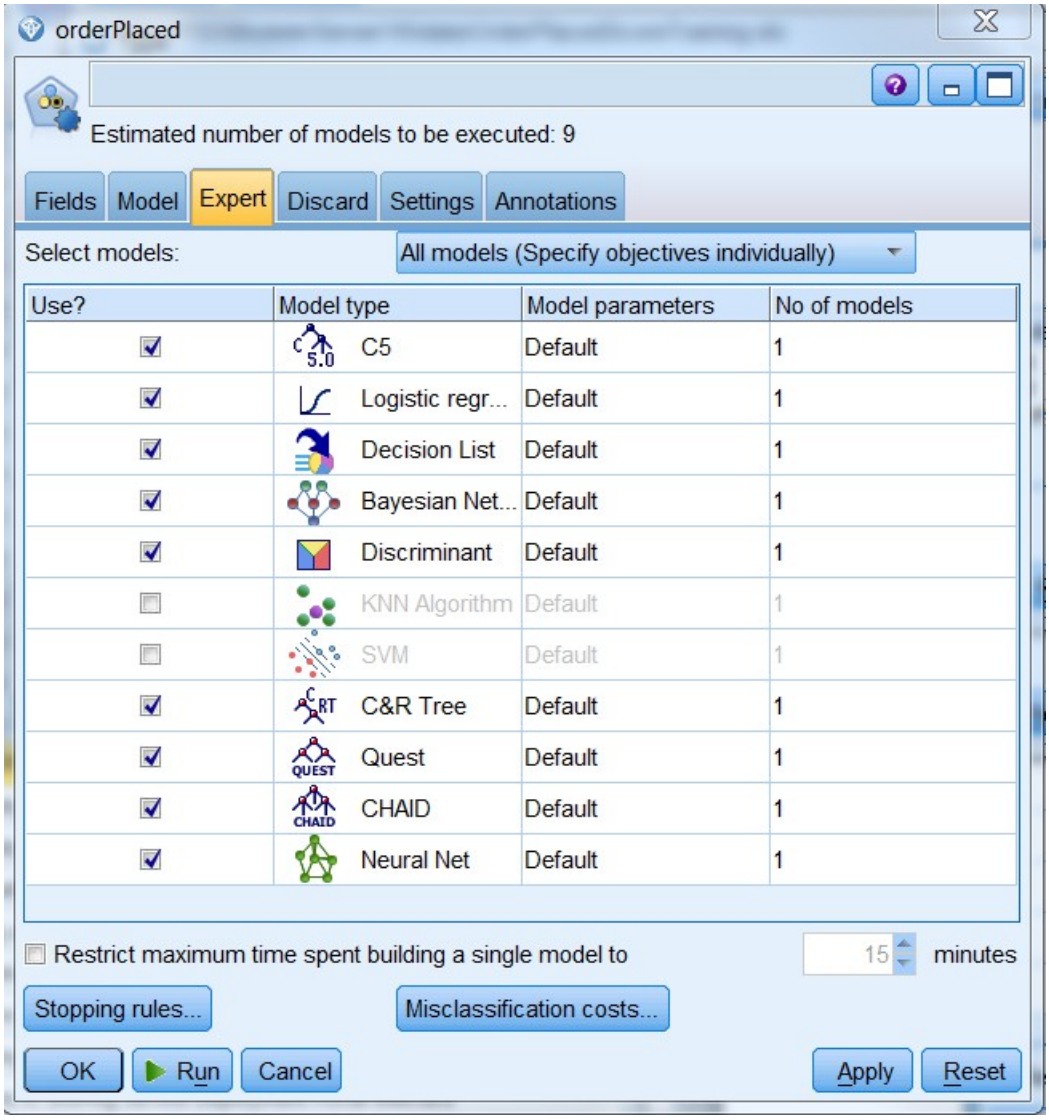
- Continuous for numbers,
- Nominal for sets of information such as strings
- Flag for selections between two options

The Values column shows the ranges of values in the data source. Both this and the Measurement values can be reassessed by clicking **Read Values**.

The Role column determines how each field should be used in later stream nodes- in this case the Auto classifier node.
Fields that are considered an input to the classifier are marked as **Input**. The field that is to be predicted is marked as **Target**.

When the model is being trained this information is passed to the orderPlaced Node in order for it to build the model.

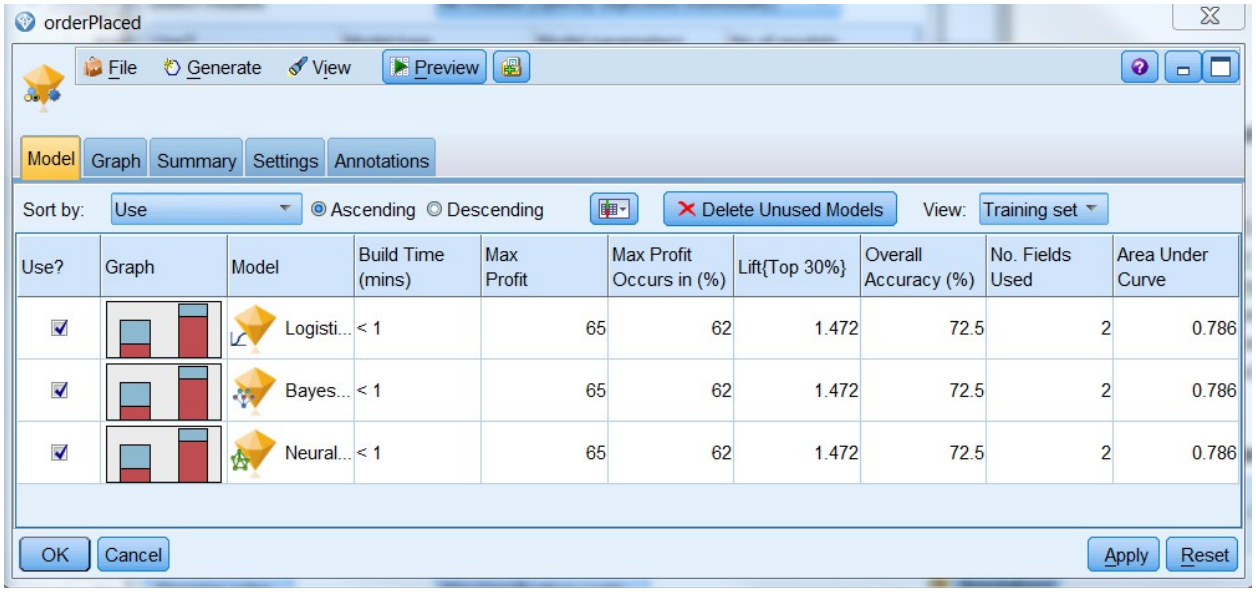
Figure 3.5. orderPlaced (Auto Classifier)



The orderPlaced Node is an Auto Classifier which will try and fit different models to the training data. The Expert tab shows the different model types that will be considered. In this case there are eight relevant models that it will try.

Clicking the **Run** button will cause the classifier to re-evaluate the training data and update the orderPlaced nugget.

Figure 3.6. orderPlaced (Nugget)

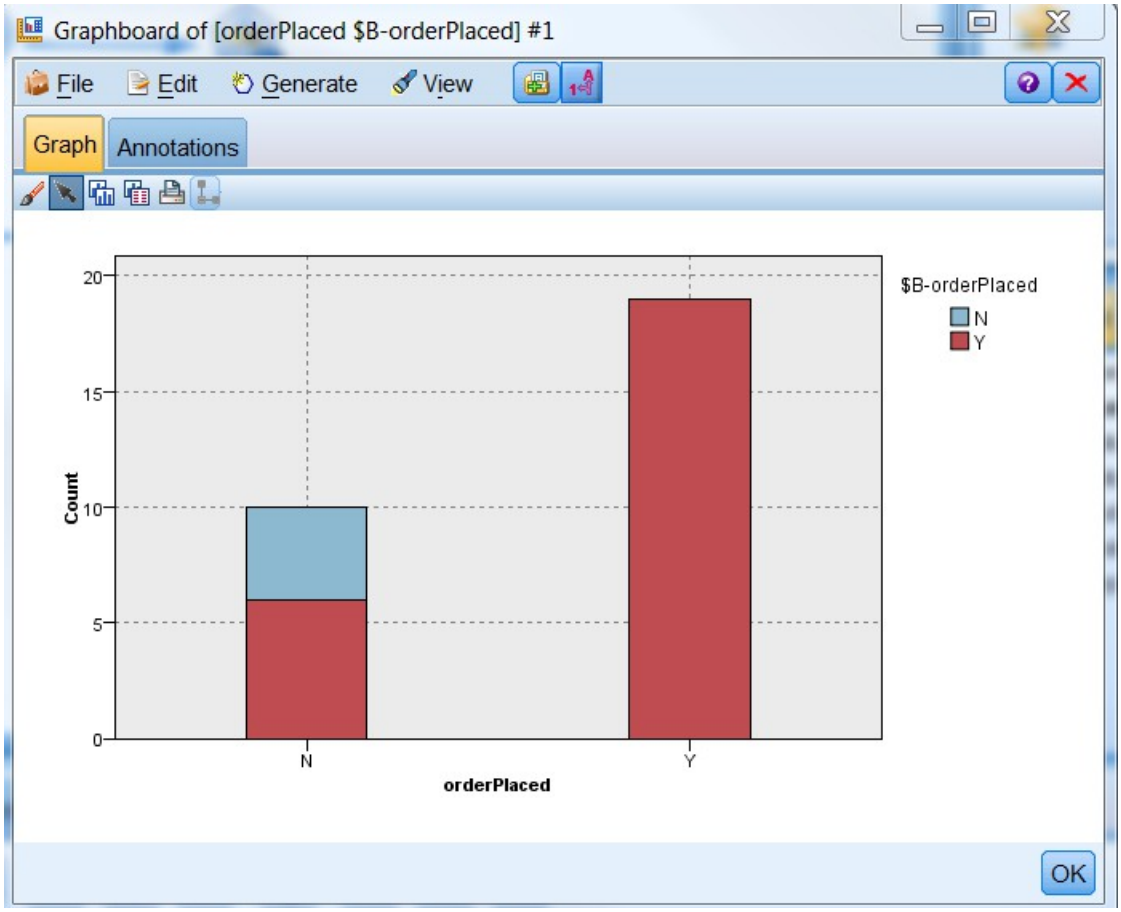


The orderPlaced nugget generated includes the three models that best fit the training data. All models have an accuracy of 72.5% and use both parameters.

For a true comparison of the predicted orderPlaced compared against the training values you can select the OrderPlacedScoring node and enable the orderPlaced field as an input to the nugget.

The accuracy of any model can be examined by clicking the Graph column in the orderPlaced nugget node for any particular model.

Figure 3.7 Graph board for Bayesian network model of orderPlaced.



This graph shows the number of predictions based on the actual value in the original data. For a more detailed comparison you can select the output table and click run. Remember to ensure that the orderPlaced value is included to ensure that the comparison can be made.

Figure 3.8 Output table orderPlaced comparison

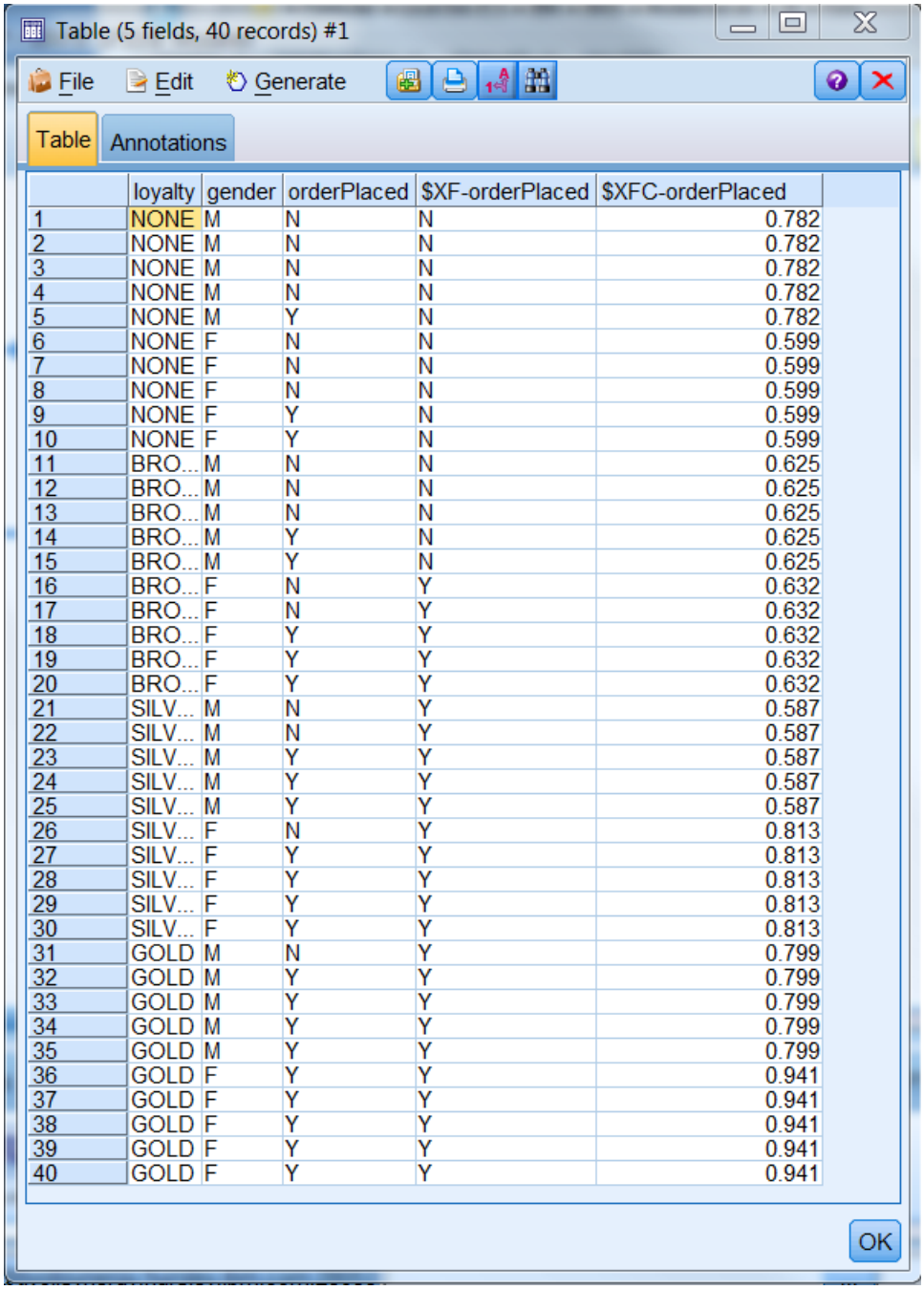


Table (5 fields, 40 records) #1

File Edit Generate

Table Annotations

	loyalty	gender	orderPlaced	$XF-orderPlaced	$XFC-orderPlaced
1	NONE	M	N	N	0.782
2	NONE	M	N	N	0.782
3	NONE	M	N	N	0.782
4	NONE	M	N	N	0.782
5	NONE	M	Y	N	0.782
6	NONE	F	N	N	0.599
7	NONE	F	N	N	0.599
8	NONE	F	N	N	0.599
9	NONE	F	Y	N	0.599
10	NONE	F	Y	N	0.599
11	BRO...	M	N	N	0.625
12	BRO...	M	N	N	0.625
13	BRO...	M	N	N	0.625
14	BRO...	M	Y	N	0.625
15	BRO...	M	Y	N	0.625
16	BRO...	F	N	Y	0.632
17	BRO...	F	N	Y	0.632
18	BRO...	F	Y	Y	0.632
19	BRO...	F	Y	Y	0.632
20	BRO...	F	Y	Y	0.632
21	SILV...	M	N	Y	0.587
22	SILV...	M	N	Y	0.587
23	SILV...	M	Y	Y	0.587
24	SILV...	M	Y	Y	0.587
25	SILV...	M	Y	Y	0.587
26	SILV...	F	N	Y	0.813
27	SILV...	F	Y	Y	0.813
28	SILV...	F	Y	Y	0.813
29	SILV...	F	Y	Y	0.813
30	SILV...	F	Y	Y	0.813
31	GOLD	M	N	Y	0.799
32	GOLD	M	Y	Y	0.799
33	GOLD	M	Y	Y	0.799
34	GOLD	M	Y	Y	0.799
35	GOLD	M	Y	Y	0.799
36	GOLD	F	Y	Y	0.941
37	GOLD	F	Y	Y	0.941
38	GOLD	F	Y	Y	0.941
39	GOLD	F	Y	Y	0.941
40	GOLD	F	Y	Y	0.941

OK

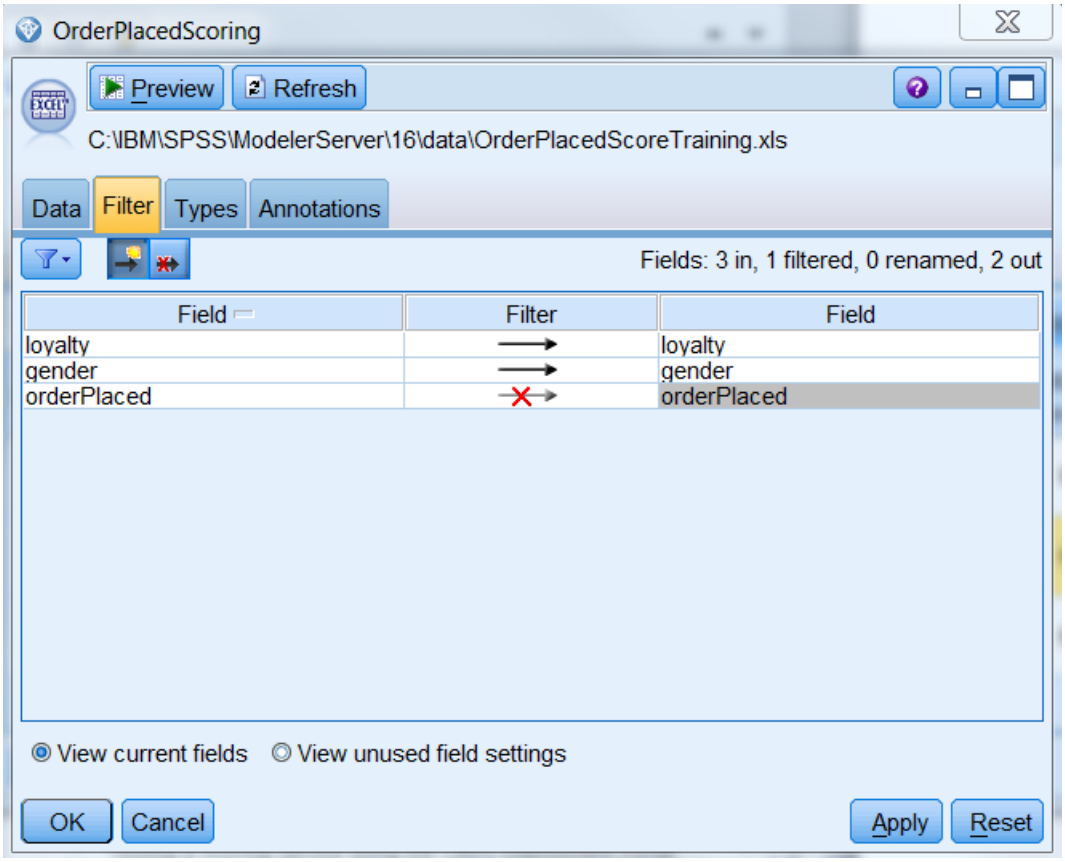
In this table the value $XF-orderPlaced indicates the predicted value and the $XFC-orderPlaced indicates the confidence of that prediction. These values are what will be returned by the scoring service.

From these results we can see that the model is not particularly accurate and would normally need to be trained with more data. The trends setup in the training information are however visible.

Predictive model development is a topic that requires careful consideration and is beyond the scope of this article. For this scenario, we will be working with the model we have here (with its limitations) and show how this can be exposed as a scoring service in SPSS.

Once you have completed your validation of the scoring service, you should ensure that the OrderPlacedScoring source filters the orderPlaced field as shown in figure 3.9 otherwise this information will be requested in the scoring service.

Figure 3.9 Filtering the target field from the scoring input



Creating and using the Scoring Service

In this section we describe how to make the OrderPlacedScoring model available as a scoring service.

The following steps need to be undertaken

- Create a Scoring Configuration in C&DS Deployment Manager to expose the OrderPlaced model as the OrderPlacedScoring Service.
- Test the OrderPlacedScoring service in the C&DS Deployment portal
- Generate a scoring service client to allow SPSS scoring services to be called from Java applications

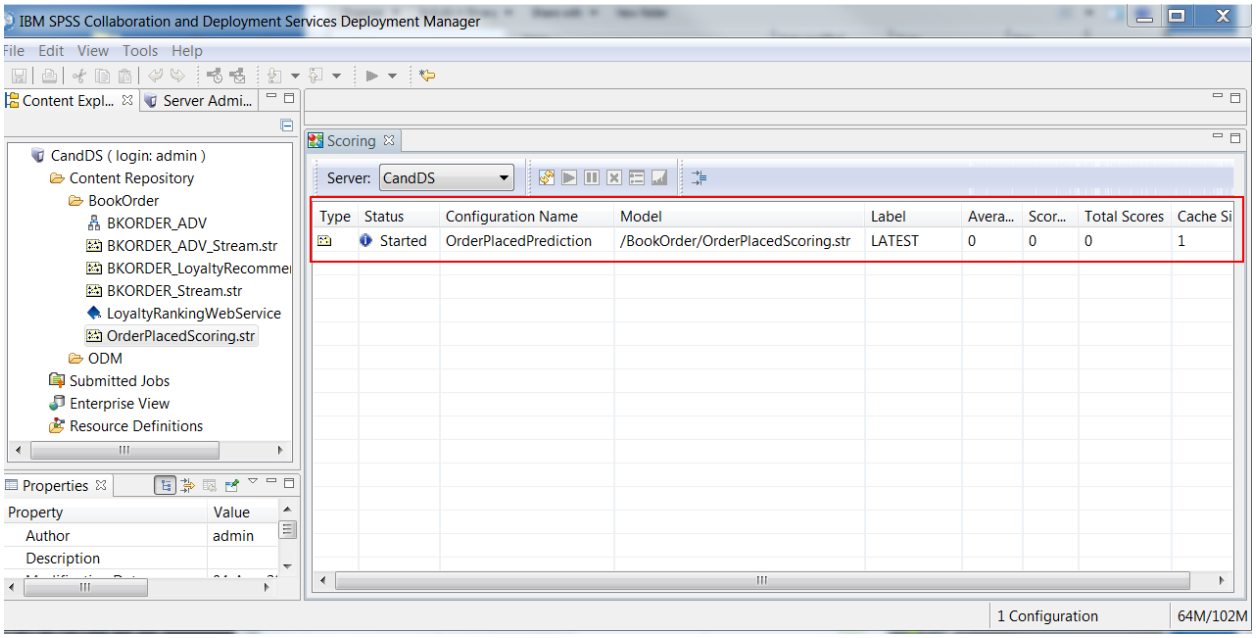
- Generate an OrderPlacedScoring service client and test program to show how the scoring service may be called by a Java application. In our scenario this will be called from IBM Integration Bus or from Business Process Manager.

To create a scoring configuration based on the OrderPlacedScoring stream:

1. Open and login to the C&DS Deployment Manager
2. Open the Content Repository, navigate to and select the **OrderPlacedScoring.str**
3. Right click and select **Configure Scoring ….**
4. In the Add New Scoring Model Configuration panel set the Name to **OrderPlacedPrediction** - This is the name we will use to identify the scoring service.
5. In **Data Provider Settings** leave the Type selected as None.
6. In the **Input Data Order** leave the order as is.
7. Leave **Input Data returned settings** unselected. In this case the input fields are returned in the output.
8. Leave **Output Data Returned Settings** as is so we can use input values score and confidence in our decisions.
9. Leave the **Logging settings** disabled.
10. In **Advanced Settings** click Finish.

This will create the Scoring configuration which can be viewed in the Scoring view. You must enable the scoring view by selecting:
View > Show View > Scoring from the menu bar.

Figure 3.10 C&DS Deployment Manager Scoring View



The scoring view shows the status of any scoring configurations and hence scoring services. It also provides the ability to stop start or delete scoring services.

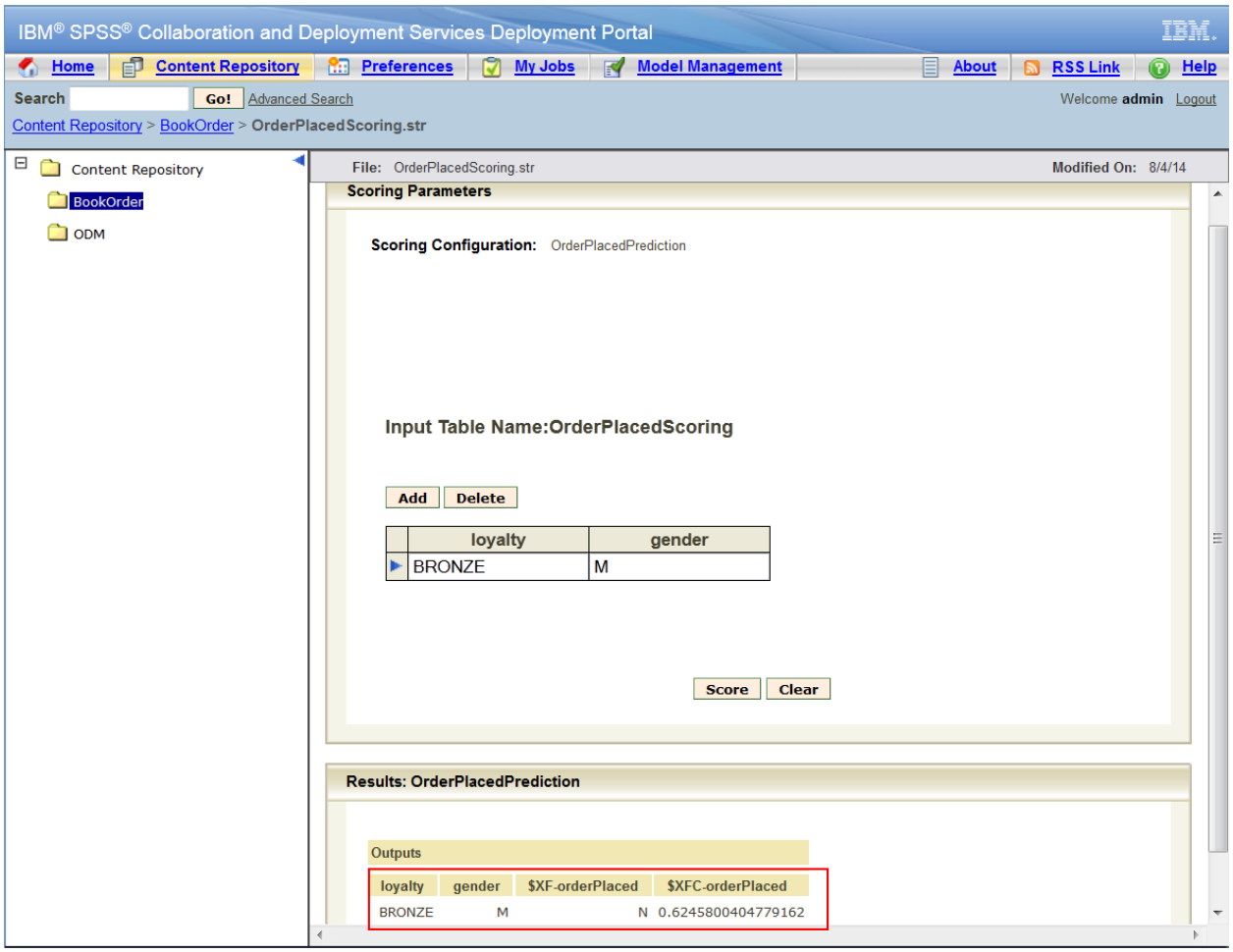
With scoring service now active, we can test it using the C&DS Deployment Portal.

The C&DS Deployment Portal is a web based application that allows you to view and manage your analytic jobs and services. Follow the following steps to test the **OrderPlacedPrediction** scoring service:

1. Open the Deployment Portal at http://localhost:9080/peb
2. Login as your user: admin/admin for our environment
3. In the Content Repository tab navigate to the BookOrder/OrderPlacedScoring.str and select it.

This opens a form that allows you to enter your scoring input parameters and obtain a score result.

Figure 3.11 Scoring Service Deployment Portal Interface



By entering various input parameters and clicking **Score** you can evaluate the predicted orderPlaced value ($XF-orderPlaced) and confidence ($XFC-orderPlaced) based on customer loyalty and gender.

This shows that the OrderPlacedPrediction scoring service is now operating using our model.

In this panel the following scoring service configuration details should be recorded as these will be needed to invoke the scoring service from a client as shown in listing 3.3.

- Scoring configuration: OrderPlacedPrediction
- Input Table Name: OrderPlacedScoring
- Input Parameters (in order): Loyalty, Gender
- Results (in order): loyalty, gender, $XF-orderPlaced, $XFC-orderPlaced

Note also that multiple scores can be obtained by adding additional rows to the input table.

The next section describes how to invoke the scoring service from a Java client.

Invoking a scoring service from a Java client

The SPSS scoring services are available through SOAP web services or through a REST API. This section describes how to develop a Java client that invokes a scoring service using SOAP web services.

The steps described are:

1. Create a JAX-WS client JAR based on the SPSS scoring services WSDL
2. Develop the Java client to connect to the scoring service
3. Develop the Java client to invoke the scoring service and obtain a loyalty score.

The C&DS documentation provides a clear description of how to create and configure a JAX-WS client for C&DS web services in the topic Web Services > JAX-WS clients. This client library then provides all classes needed to invoke the scoring service remotely.

The key steps are:

1. Open a command prompt in the directory that you wish to create the client.
2. Invoke the Java 6 wsimport.exe command referencing the scoring service wsdl:
   `<JAVA6_HOME>\bin\wsimport.exe`
   `http://localhost:9080/scoring/services/Scoring.HttpV2?wsdl`
3. Package the classes produced into a jar by issuing the command:
   `<JAVA6_HOME>\bin\jar.exe -cvf SPSSscoring.jar *`
4. Include the SPSSscoring.jar in your Java client project libraries.

With the scoring service client library, it is fairly easy to create a Java client project.
First create the OrderPlacedScoringClient Project and add the SPSSscoring.jar to the build class path.

These code segments describe the essential parts of the project.

Listing 3.1 Loyalty Scoring Client Class and key properties

```
public class OrderPlacedScoringClient {
      String scoring URL =
        "http://localhost:9080/scoring/services/Scoring.HttpV2";
```



```
String username = "admin";
String password = "admin";
String predictedOrderPlaced = "Y";
double predictedConfidence = 0.0;
ScoringServices service = null;
public OrderPlacedScoringClient() {
      super();
      connect();
}
```

The client class provides three properties used to connect to the scoring service:

- **scoringURL** identifies the remote url of the scoring service
- **username** and
- **password** identify the credentials needed to connect to the server.

The response from the scoring service is held in two properties:

- **predictedOrderPlaced** holds the expected response of the customer
- **predictedConfidence** hold the confidence that the scoring service places on that response.

The **service** property references the ScoringServices class created by the JAX-WS utility and is established by the connect method shown in listing 3.2. The connection then allows the scoring calls to be made as shown in listing 3.3.

Listing 3.2 – Connecting to a scoring service

```
//Establish connection to the scoring service
private void connect() {
   try {
      //Create the main service connection
      service = new ScoringServices(
            new URL(scoringURL+"?WSDL"),
            new QName("http://xml.spss.com/scoring/wsdl","ScoringServices"));
      //set the security handler
      service.setHandlerResolver(new HandlerResolver()
      {
      @Override
      public List<Handler> getHandlerChain(PortInfo portInfo)
            {
            List<Handler> handlerChain = new ArrayList<Handler>();
            handlerChain.add(
                  new SecurityHandler(
                        username, password,
                        "en-US;q=1.0, en;q=0.8"));
                  return handlerChain;
            }
      });
      //Define the service endpoint
      ScoringV2 serviceEndpoint = service.getHttpV2();
      Map<String, Object> requestContext =
            ((BindingProvider)serviceEndpoint).getRequestContext();
```



```
        requestContext.put(
                BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                scoringURL);
    } catch (MalformedURLException e) {
                e.printStackTrace();
    }
}
```

This code performs the following steps:

- Create the ScoringServices proxy using the scoring service url. This obtains the WSDL at runtime from the endpoint.
- Setup a handler to resolve security challenges. See SoapHandler example for more details on this. The handler is supplied with the credentials.
- Define the service endpoint. Note that this example uses the V2 version of the scoring service APIs. You must make sure that you use V2 classes consistently throughout the Java client.

Sample code to invoke the scoring service is shown in listing 3.3. You need to have access to the scoring service configuration details recorded from figure 3.11.

Listing 3.3 – Invoking the scoring service

```
String getScore(String loyalty, String gender) {
        //Create a new score request
        ScoreRequest scoreRequest = new ScoreRequest();
        //Define the OrderPlacedPrediction configuration
        scoreRequest.setId("OrderPlacedPrediction");
        //Create the table and first row
        RequestInputTable rInputTable = new RequestInputTable();
        rInputTable.setName("OrderPlacedScoring");
        RequestInputRow rInputRow = new RequestInputRow();
        //Add the row and table to the request
        rInputTable.getRequestInputRow().add(rInputRow);
        scoreRequest.getRequestInputTable().add(rInputTable);
        //Setup the parameters
        Input loyaltyi = new Input();
        loyaltyi.setName("loyalty");
        loyaltyi.setValue(loyalty);
        rInputRow.getInput().add(loyaltyi);
        Input genderi = new Input();
        genderi.setName("gender");
        genderi.setValue(gender);
        rInputRow.getInput().add(genderi);
        //Invoke the scoring service
        ScoreResult scoreResult = null;
        try {
                scoreResult = service.getHttpV2().getScore(scoreRequest);
        } catch (MissingDataException | ScoringException e) {
                e.printStackTrace();
        }
        //Get the first Row of values
```



```
        RowValues values = scoreResult.getRowValues().get(0);
        //Get the predicted loyalty column 3
        predictedOrderPlaced = values.getValue().get(2).getValue();
        //get the confidence column 4
        predictedConfidence =
           Double.valueOf(values.getValue().get(3).getValue());
        //return the results
        System.out.println("Predicted orderPlaced: " + predictedOrderPlaced);
        System.out.println("Predicted Confidence: " + predictedConfidence);
        return predictedOrderPlaced;
}
```

This code segment performs the following steps:

- A new scoring request is created for the **OrderPlacedPrediction** scoring configuration
- The input table **OrderPlacedScoring** is created and an initial row added
- The input parameters **loyalty** and **gender** are added to the row
- The scoring service is invoked using the getScore() operation
- The first (and only) row of results are obtained
- The **predictedOrderPlaced** and **predictedConfidence** values are extracted from the row based on their position
- The results are printed and returned.

This routine can then be setup in a test harness or invoked from an application to obtain the scores as shown in listing 3.4.

Listing 3.4 Test Harness Program

```
public static void main(String[] args) {
       OrderPlacedScoringClient client = new OrderPlacedScoringClient();
       System.out.println("***********************************************");
       System.out.println("Prediction for: NONE M : " +
              client.getScore("NONE","M") );
       System.out.println("***********************************************");
       System.out.println("Prediction for: BRONZE M : " +
             client.getScore("BRONZE","M") );
       System.out.println("***********************************************");
       System.out.println("Prediction for: SILVER M : " +
             client.getScore("SILVER","M") );
       System.out.println("***********************************************");
       System.out.println("Prediction for: GOLD M : " +
             client.getScore("GOLD","M") );
       System.out.println("***********************************************");
       System.out.println("Prediction for: NONE F : " +
             client.getScore("NONE","F") );
       System.out.println("***********************************************");
       System.out.println("Prediction for: BRONZE F : " +
             client.getScore("BRONZE","F") );
       System.out.println("***********************************************");
       System.out.println("Prediction for: SILVER F : " +
             client.getScore("SILVER","F") );
```



```
        System.out.println("***********************************************");
        System.out.println("Prediction for: GOLD F : " +
              client.getScore("GOLD","F") );
}
```

This simple test harness invokes the scoring service client for a number of different **loyalty** and **gender** combinations providing results shown in listing 3.5.

Listing 3.5 Test Harness Results

```
*********************************************
Predicted orderPlaced: N
Predicted Confidence: 0.7824129100175884
Prediction for: NONE M : N
*********************************************
Predicted orderPlaced: N
Predicted Confidence: 0.6245800404779162
Prediction for: BRONZE M : N
*********************************************
Predicted orderPlaced: Y
Predicted Confidence: 0.586613909192799
Prediction for: SILVER M : Y
*********************************************
Predicted orderPlaced: Y
Predicted Confidence: 0.7990290299353644
Prediction for: GOLD M : Y
*********************************************
Predicted orderPlaced: N
Predicted Confidence: 0.5987441242914456
Prediction for: NONE F : N
*********************************************
Predicted orderPlaced: Y
Predicted Confidence: 0.6322010716239493
Prediction for: BRONZE F : Y
*********************************************
Predicted orderPlaced: Y
Predicted Confidence: 0.8133479704999215
Prediction for: SILVER F : Y
*********************************************
Predicted orderPlaced: Y
Predicted Confidence: 0.9408322523714858
Prediction for: GOLD F : Y
```

This shows how the scoring service can be invoked from Java. The same pattern can be applied when integrating into IBM Integration Bus using a Java Compute node or into Business Process Manager using an integration service. This means that the scoring service may now be easily integrated into an application to provide the insight the decision service needs to improve the operational decision making.

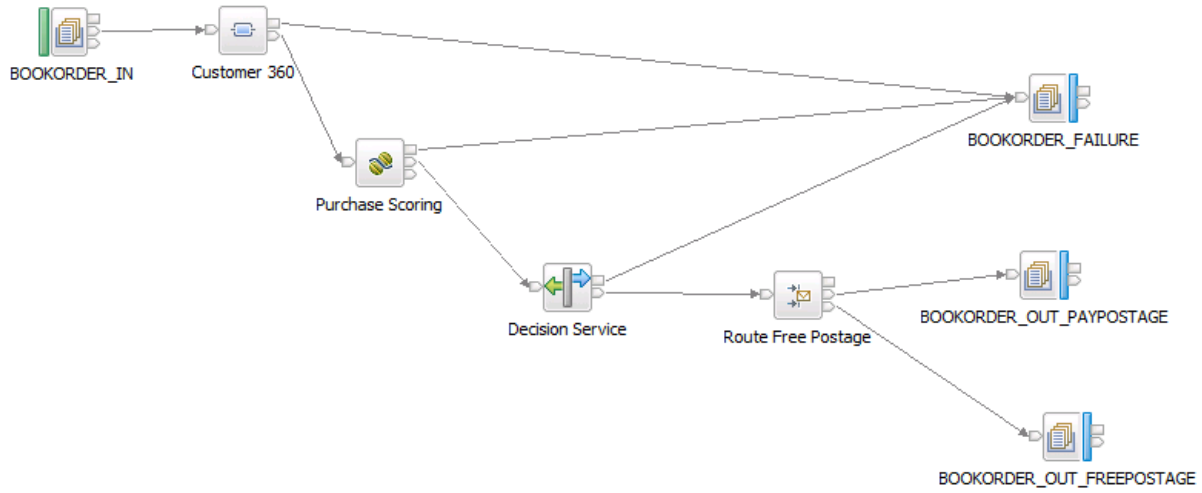
Pattern 4 – IBM Integration Bus Book Order Application

This pattern shows how analytics can be combined with operational decision management techniques in an IBM Integration Bus message flow and thus applied to optimize incoming book order transactions.

Book Order Application Message Flow

IBM Integration bus provides a flexible means of augmenting the information needed to make an informed pricing decision through the use of a variety of processing nodes in a message flow as shown below.

Figure 4.1 Hybrid Book Order Application Message Flow



Quote request messages arrive on a BOOKORDER_IN queue and their arrival initiates the flow. The *bookOrder* message is parsed from XML allowing access to the attributes and elements defined in the book order schema. This schema has been defined in ODM and imported into IIB from the ruleset that will be executed in the decision service node. It defines elements for the *bookOrder*, *customer* and *score* . These define the elements that will need to be passed to the decision service to make the pricing decision.

The **Customer 360** node is a Mapping node that extracts the customerID from the *bookOrder* message and performs a database lookup to retrieve the customer 360 degree information. This information is added to a *customer* element which is placed in the local environment for the message.

The **Purchase Scoring** node is a JavaCompute node that extracts the customer loyalty and gender from the *customer* element and performs a call to the SPSS scoring service to retrieve the likelihood that the customer will eventually place an order. This information is placed in a *score* element in the local environment.

The **Decision Service** node then takes the *bookOrder*, *customer*, and *score* elements and calculates the order price and discounts using a call to the BookOrderPricing ruleset deployed on the JSE RES configurable service provided by IBM Integration Bus.

The **Route Free Postage** node then routes the message to either the BOOKORDER_OUT_PAYPOSTAGE queue (if post and packing is greater than 0) or the BOOKORDER_OUT_FREEPOSTAGE (if post and packing is zero).
In the event of any failures or exceptions the bookOrder message is riuted to a BOOKORDER_FAILURE queue.

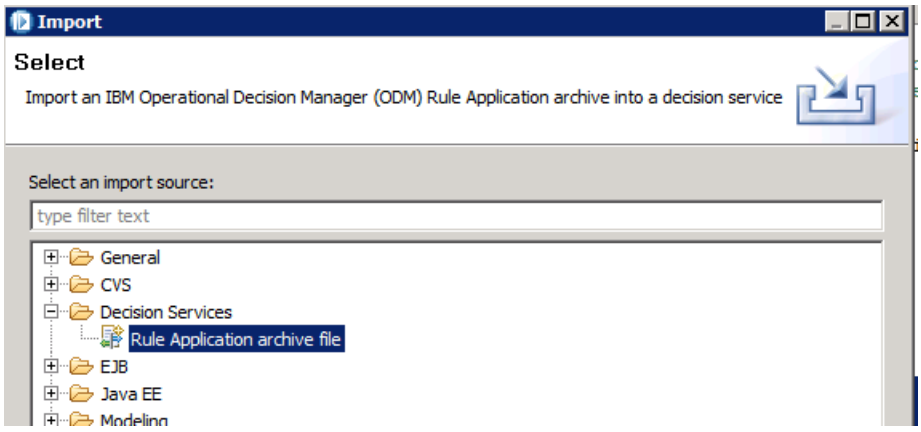
The following sections describe the three main nodes used in this flow.

Decision Service Node

The decision service node is based on the BookOrderDecisionServiceV2 imported from ODM. The V2 discriminator in the name is used to ensure that the IIB runtime does not confuse this version with earlier versions of the decision service that use different schemas.

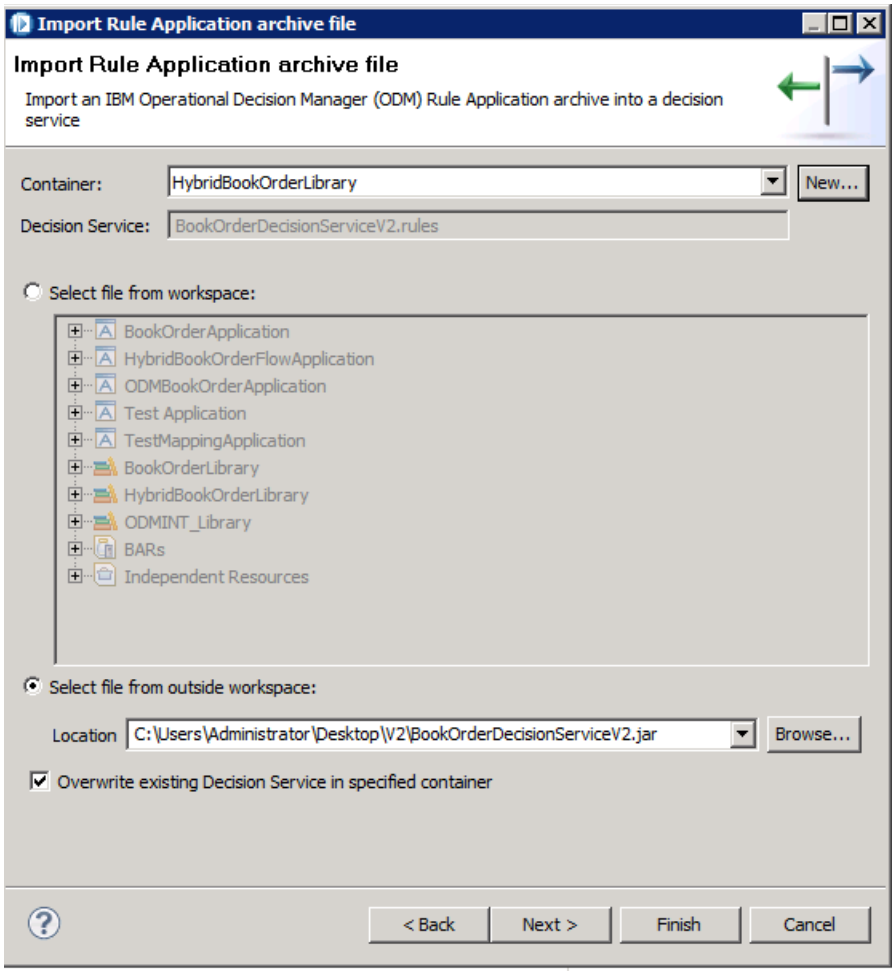
The steps undertaken to generate this decision service are as follows:

Figure 4.2 Import the decision service into the integration project



You first start the wizard to import an ODM RuleApp archive into an integration project.

Figure 4.3 Select the RuleApp archive file containing the ruleset

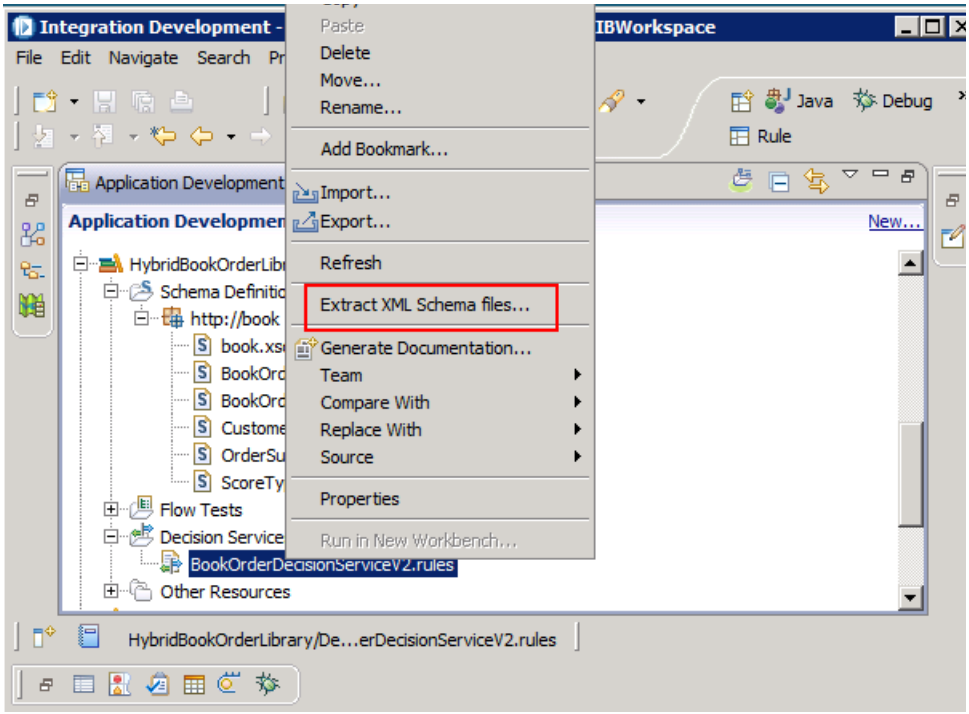


You first create a library to hold the decision service. In this case it is called *HybridBookOrderLibrary.*

To refresh the interface or local implementation to a decision service, the definition may be overwritten by selecting the checkbox. You can see the rulesets that will be imported and those that cannot be imported due to incompatibility with IIB before clicking **Finish**.

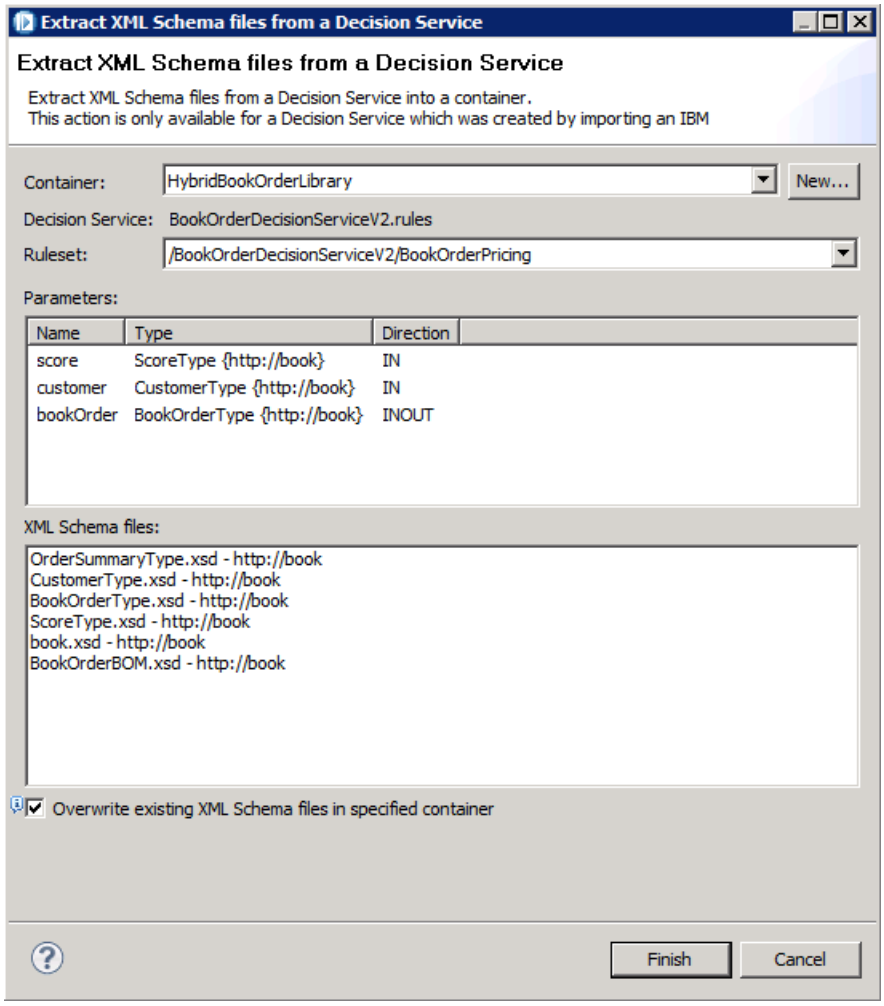
To import the schemas so they can be used in the message flow select the imported decision service (BookOrderDecisionServiceV2.rules file) and right-click **Extract XML schema files.**

Figure 4.4 Importing schemas from a ruleset



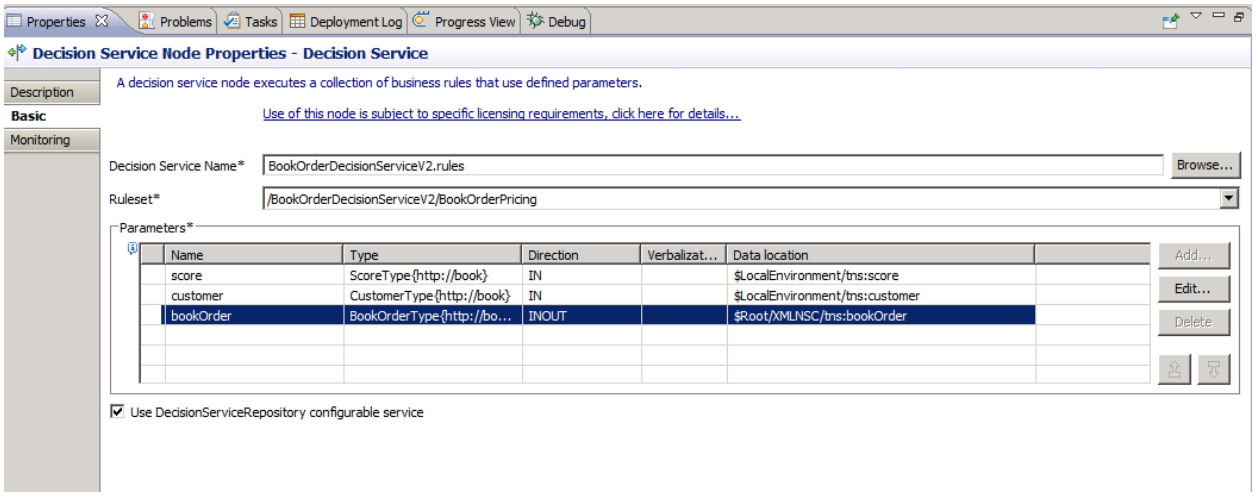
This provides a similar page that allows you to override existing schema files within the project.

Figure 4.5 Updating schemas from a ruleset



When the schemas are configured the decision service node can be created and configured in the Message Flow using the properties tab.

Figure 4.6 Decision Service Node properties

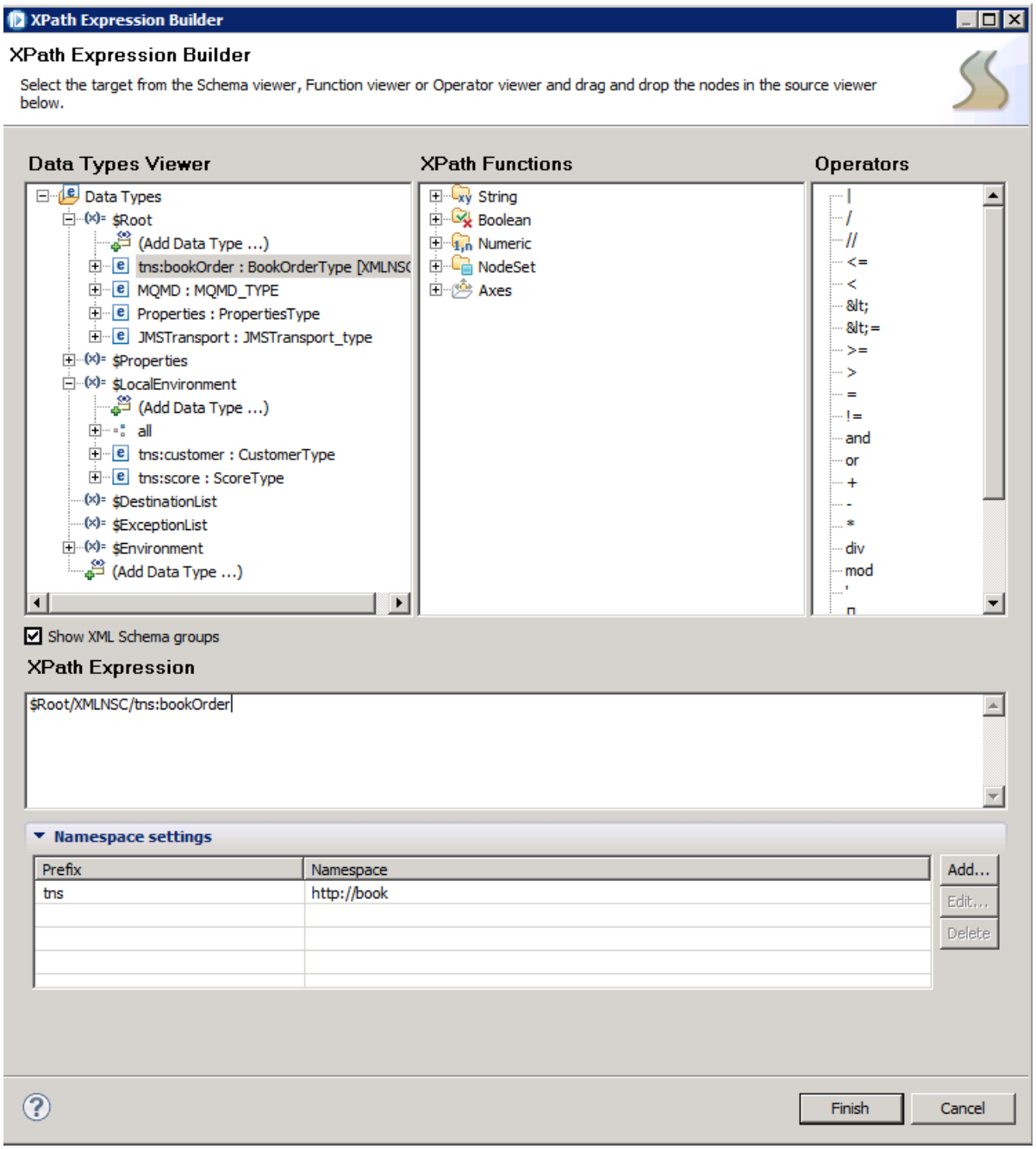


In this configuration we can see that BookOrderDecisionServiceV2/BookOrderPricing ruleset that we imported has been selected and that the node should use the DecisionServiceRepository configurable service. This means that the rules can be updated and redeployed from ODM without having to redeploy the message flow.

If you are using a configurable decision service repository, you should ensure that a corresponding ruleset is deployed to that repository so that it can be loaded at runtime. The ruleset path needs to match the path shown in the ruleset field. IIB is not specific about ruleset versions and so will invoke the latest version of the ruleApp and ruleset. This is why incompatible rulesets should make the ruleApp path distinct from each other.

The parameters passed to the ruleset need to be mapped to variable in the message tree and local environment. In this case the bookOrder is INOUT and will return the quoted price and discounts while the customer and score parameters are read only. Each of these parameters is then mapped to a data location that is recognized within the message flow. For the score and customer, this will come from the local environment while for the bookOrder the parameter will come from the message body. These mappings can be selected using the XPath expression builder.

Figure 4.7 XPath Expression Builder mapping of parameter values



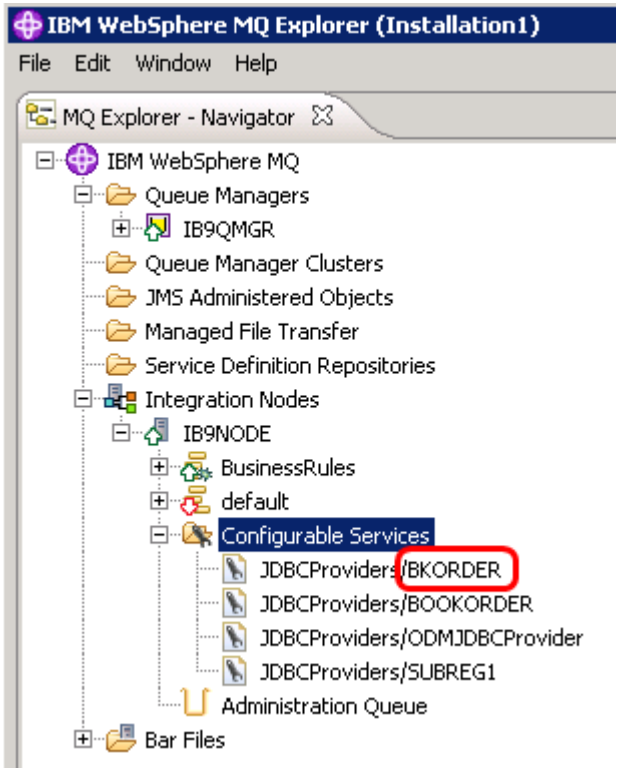
With the parameters mapped the decision service node is complete.

Customer 360 Node

The main goal of this node is to perform a query on the BKORDER database provided by the analytics and retrieve the best information about the customer. Several approaches to this are

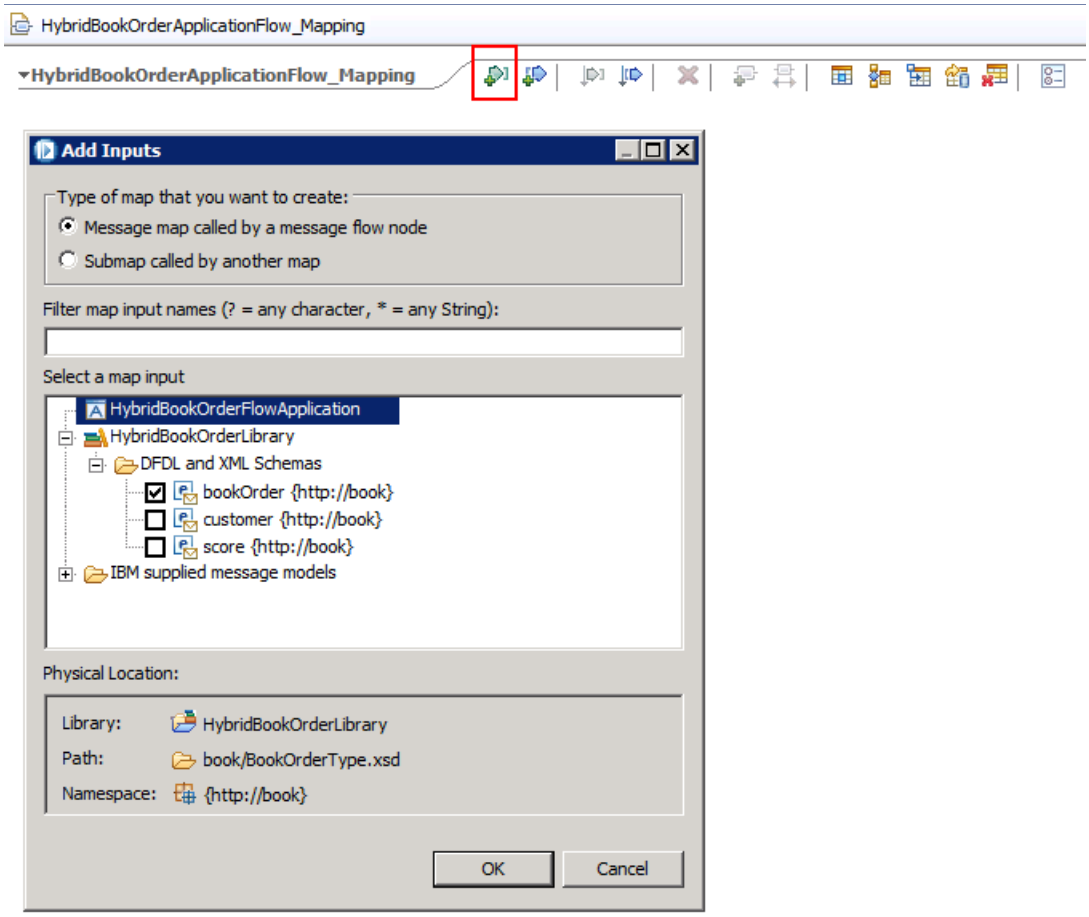
possible but in this article we will use a graphical mapper node which provides an easy way of retrieving the information from the database without requiring knowledge of Java or SQL.
In order for the mapper node to access a database, a JDBCProvider configurable service needs to be setup as shown in figure 4.8.

Figure 4.8 JDBCProviders configurable service for BKORDER



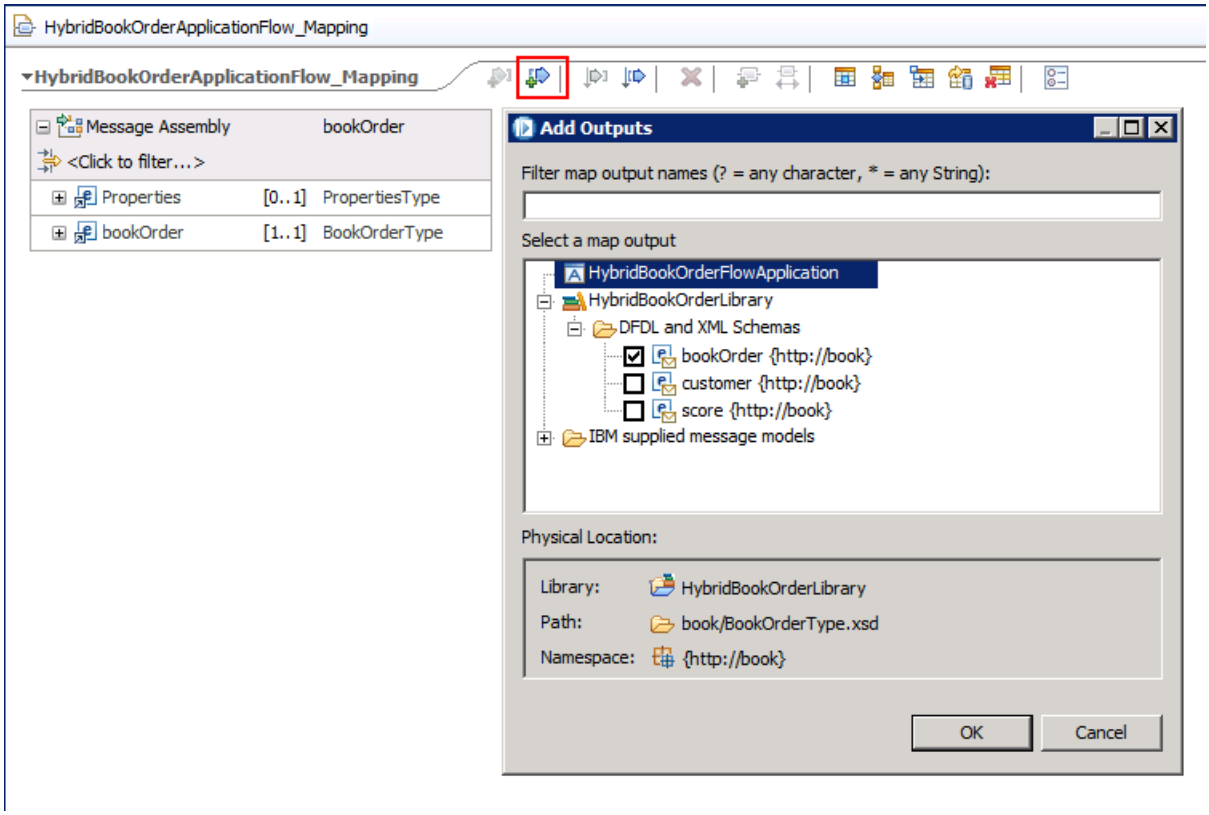
You first create a mapper node in the flow which brings up the mapper node wizard. This can be undertaken in the wizard or in the mapping editor. The first step is to define the input to the map (the bookOrder message) by clicking the "Add Input" icon as shown in Figure 4.8.

Figure 4.9 Mapping Editor add input



You should select the bookOrder schema element as the main message to be mapped and click **OK**. All we require from this is the customerID but the message must be copied across to the output of the mapper which should be defined by clicking the add output icon as shown in figure 4.10.

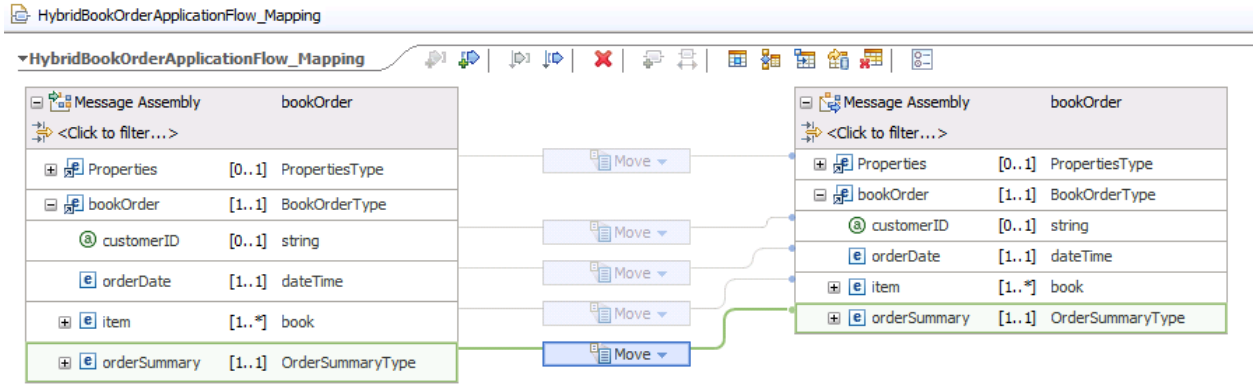
Figure 4.10 Mapper Editor Output definition



Click **OK** to add the output for the mapper.

You must now map the elements of the bookOrder across from the input to the output as shown in figure 4.11. It is important that you map the sub elements of bookOrder rather than the bookOrder itself otherwise the message cannot be serialized back into XML correctly for the decision service call.

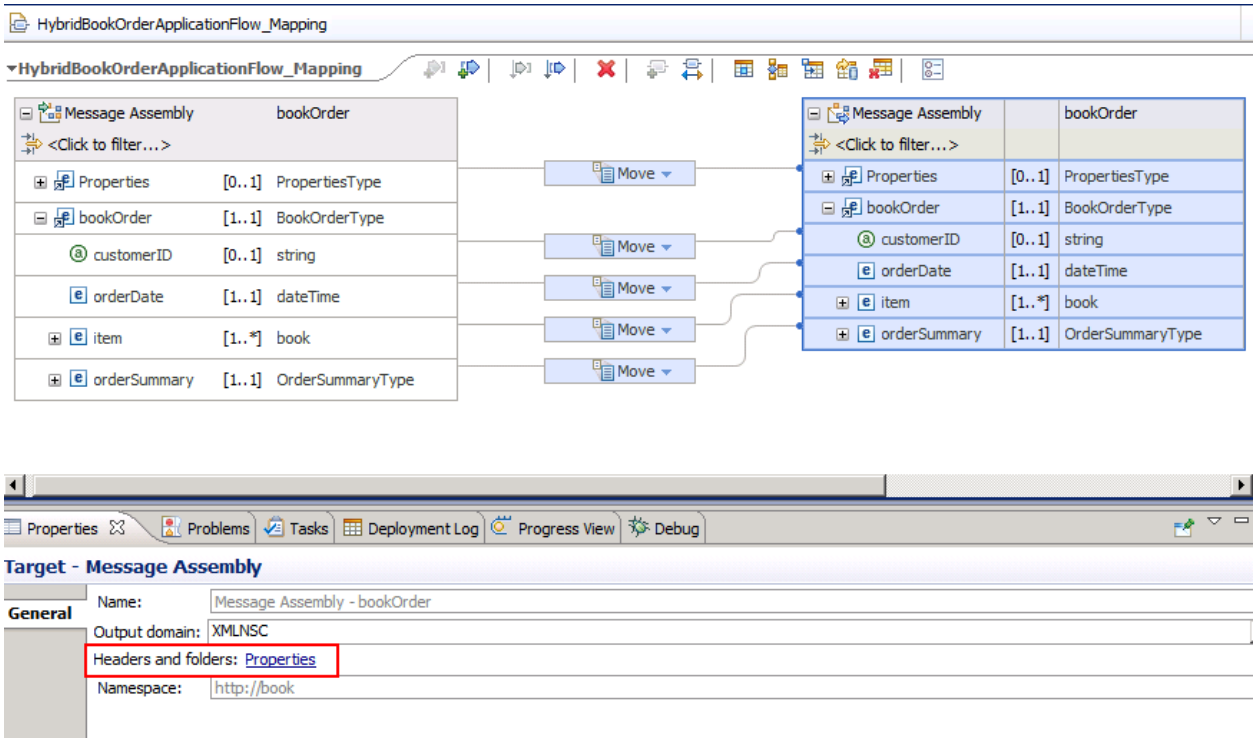
Figure 4.11 Mapper message element mapping



To perform the mapping expand the bookOrder element on both sides and drag from the input sub element to the corresponding output element. This should result in a Move task being created for each sub element.

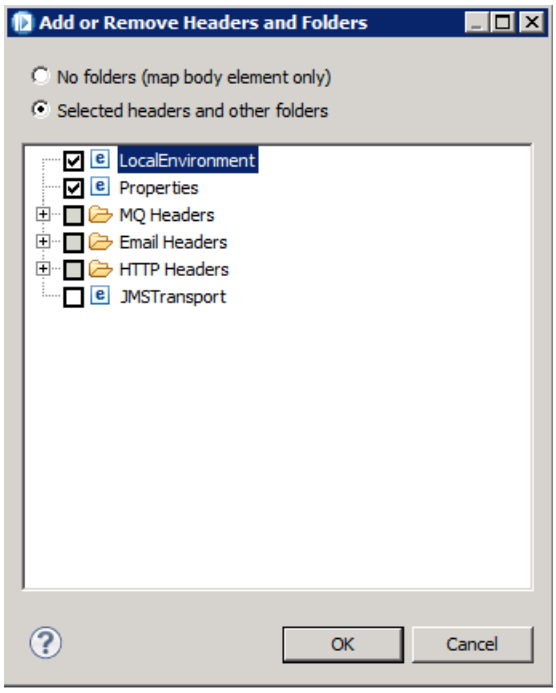
You now need to create the customer element in the local environment for the output. The local environment is not included by default so you need to add it by selecting the output message assembly and clicking the Headers and folders link which normally just includes *Properties* as shown in Figure 4.12.

Figure 4.12 Modifying Target Message assembly headers and folders.



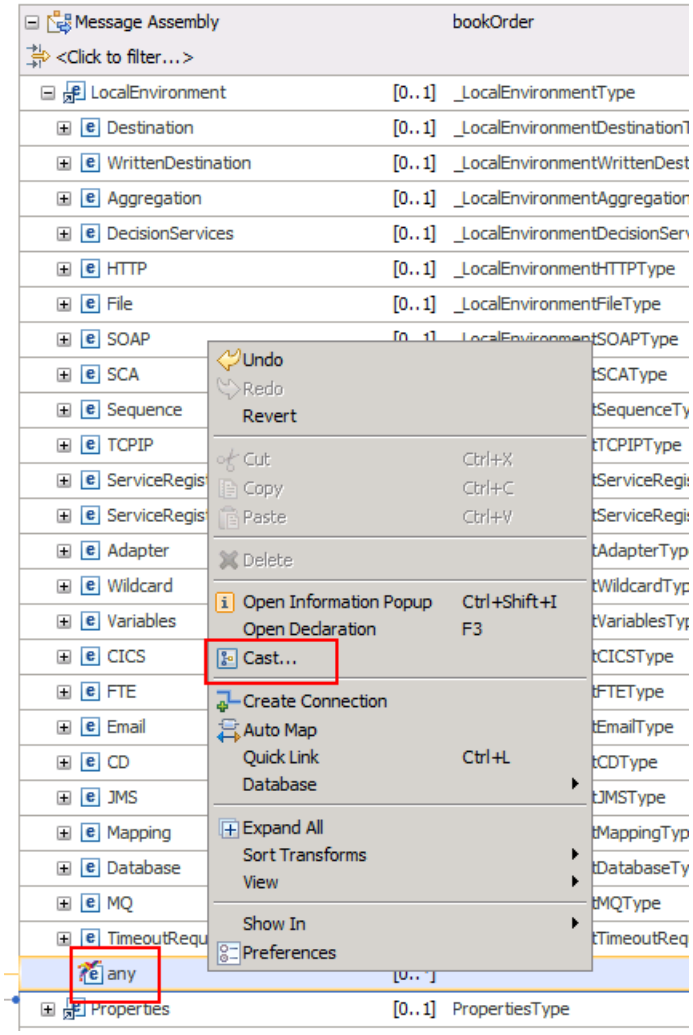
In the pop up that results select the LocalEnvironment check box as shown in figure 4.13.

Figure 4.13 Adding a Local Environment to the target message assembly.



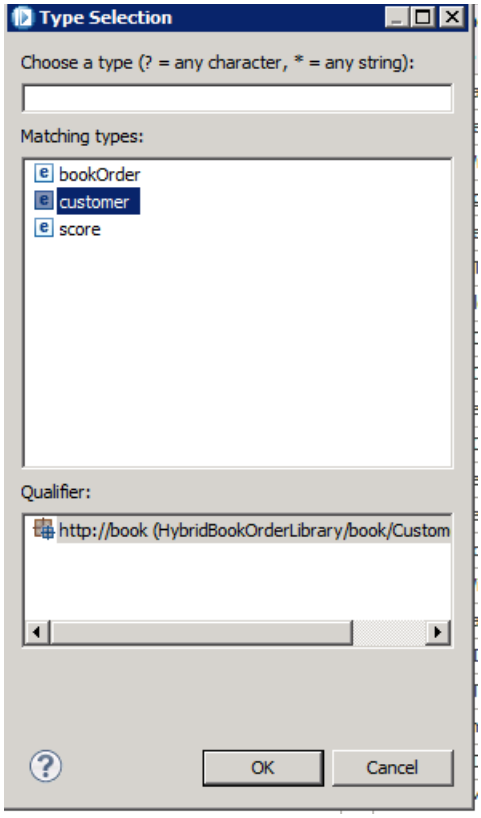
Click OK to add the LocalEnvironment to the assembly. You now need to add the *customer* element to the local environment. The local environment supports an "any" element so you need to select that and add a cast to customer as shown in figure 4.14.

Figure 4.14 Casting an element type in the Local Environment.



In the **Type Selection** pop-up select the *customer* element as shown in figure 4.15.

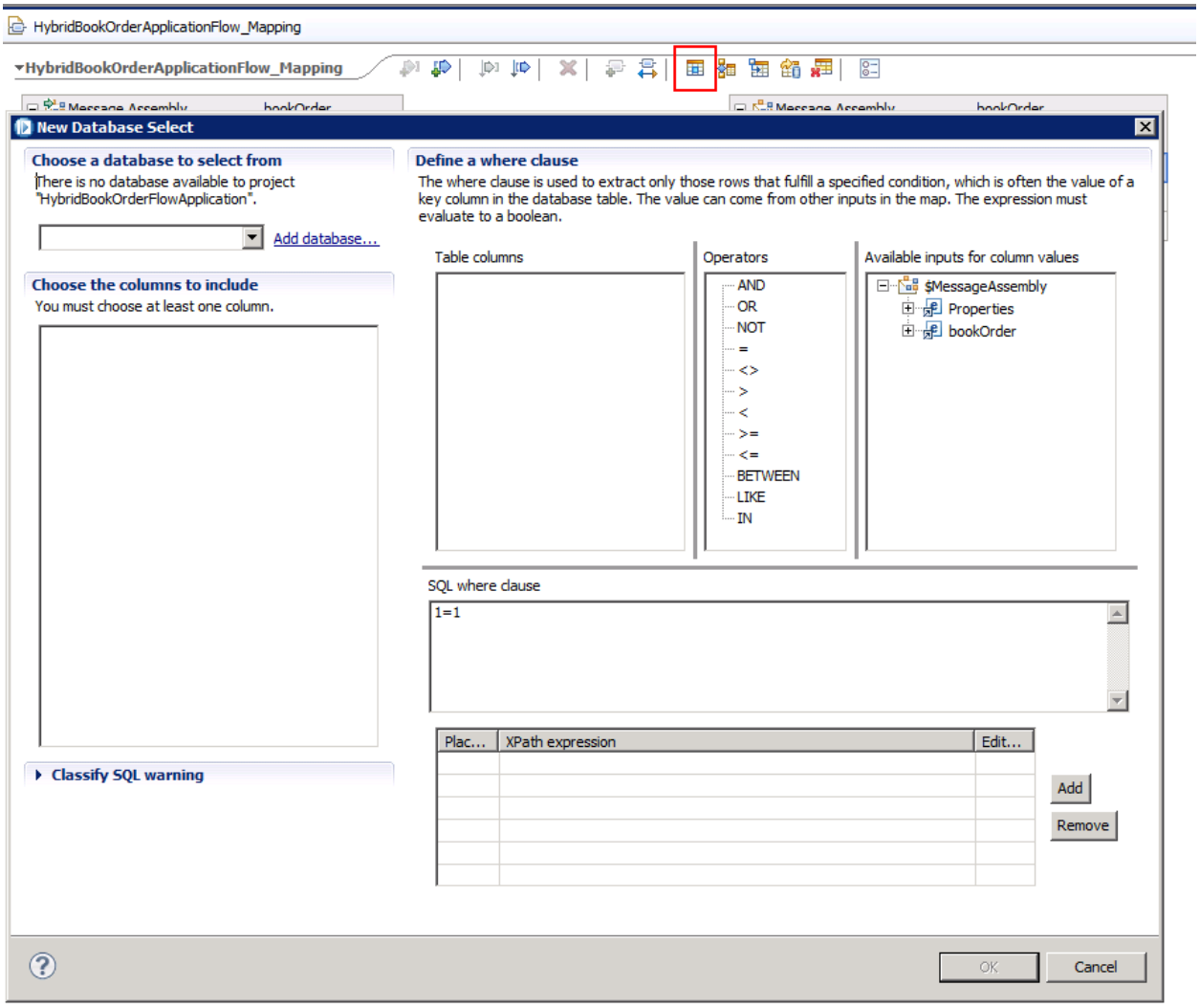
Figure 4.15 Selecting a type for a Local Environment variable.



Click **OK** and the customer element should be visible in the LocalEnvironment. You will need this when you map the results of the database query.

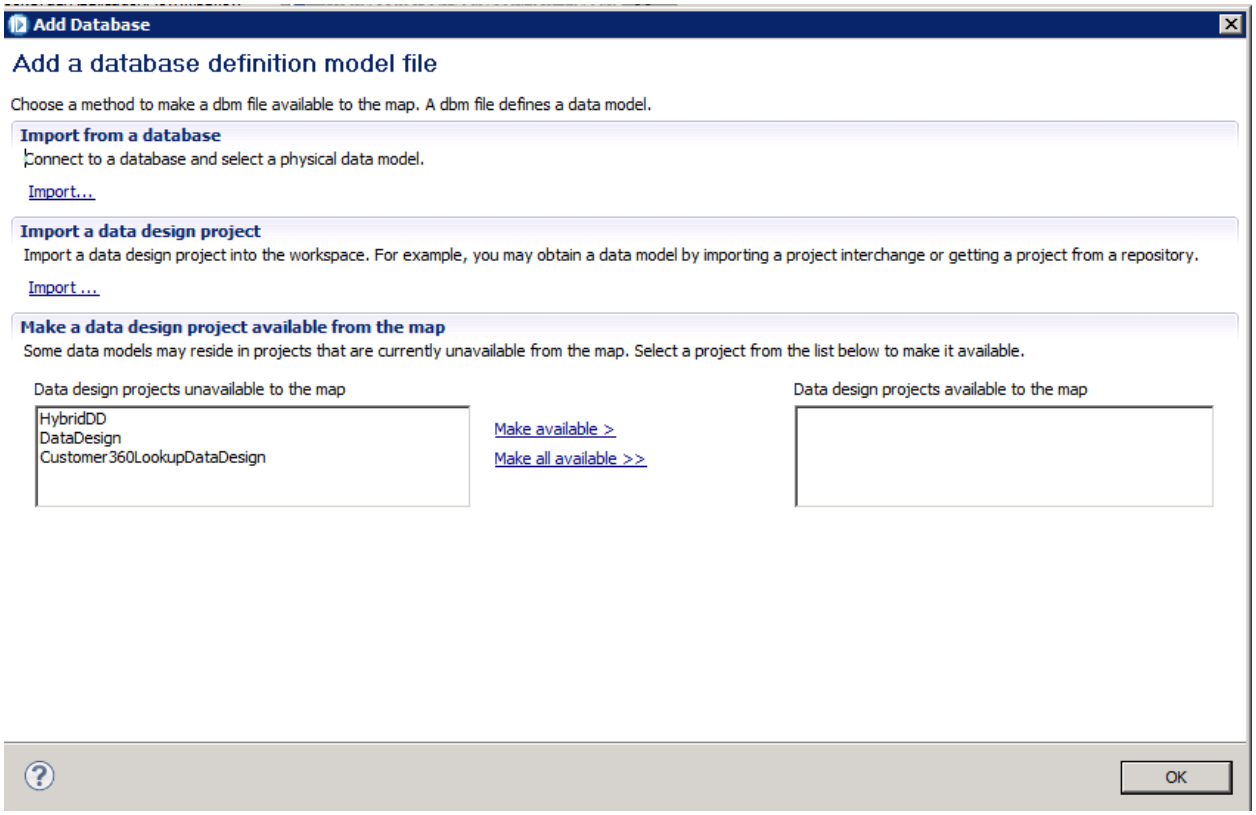
The next step is to add a "select rows from database" to the mapping by clicking the icon as shown in figure 4.16.

Figure 4.16 Mapper Database select editor.



The first step is to define the database. This is undertaken by clicking the Add database link in the editor. The Add a database definition model file appears as shown in figure 4.17.

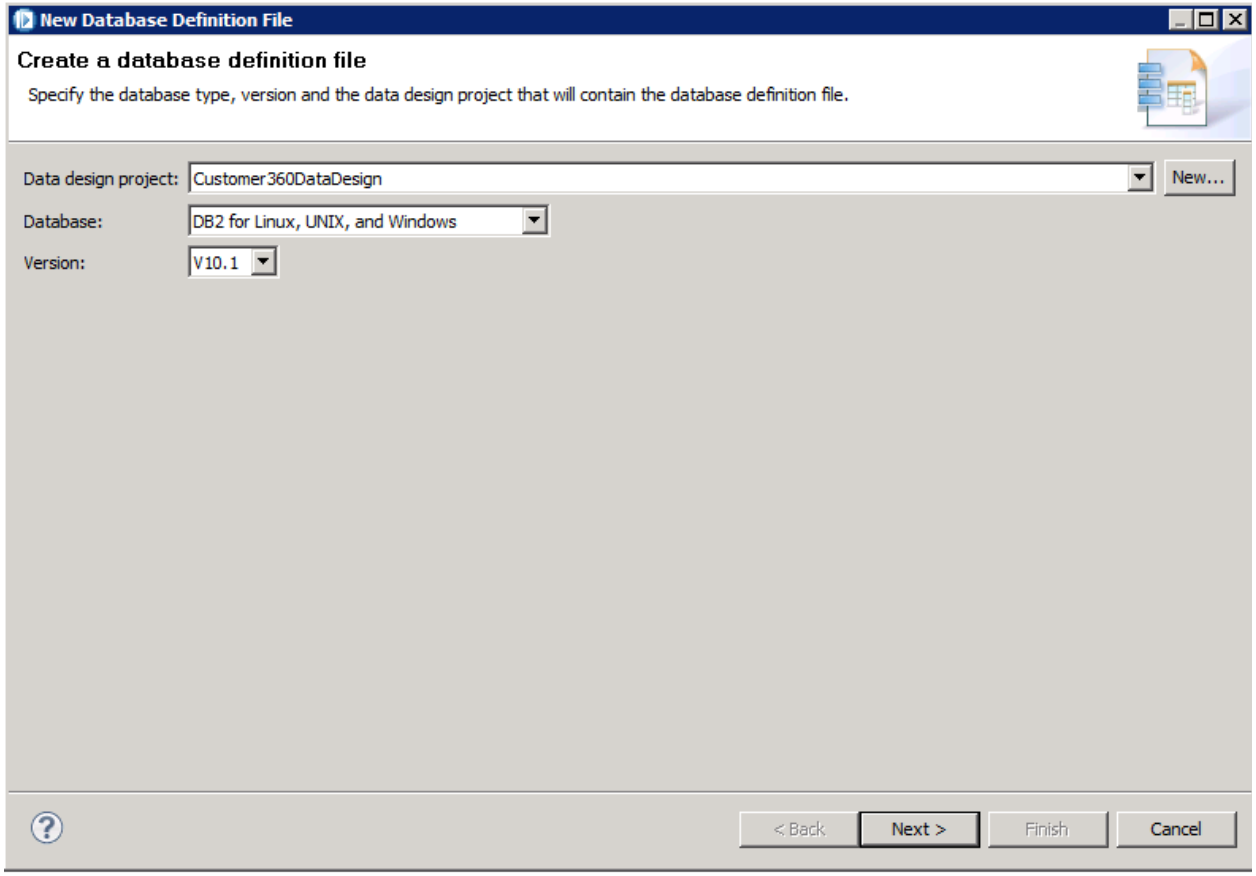
Figure 4.17 Database definition model.



This screen allows a number of options depending on whether you already have a Data Definition project set-up or not. We will assume in this article that you have access to a database but do not yet have a data design project.

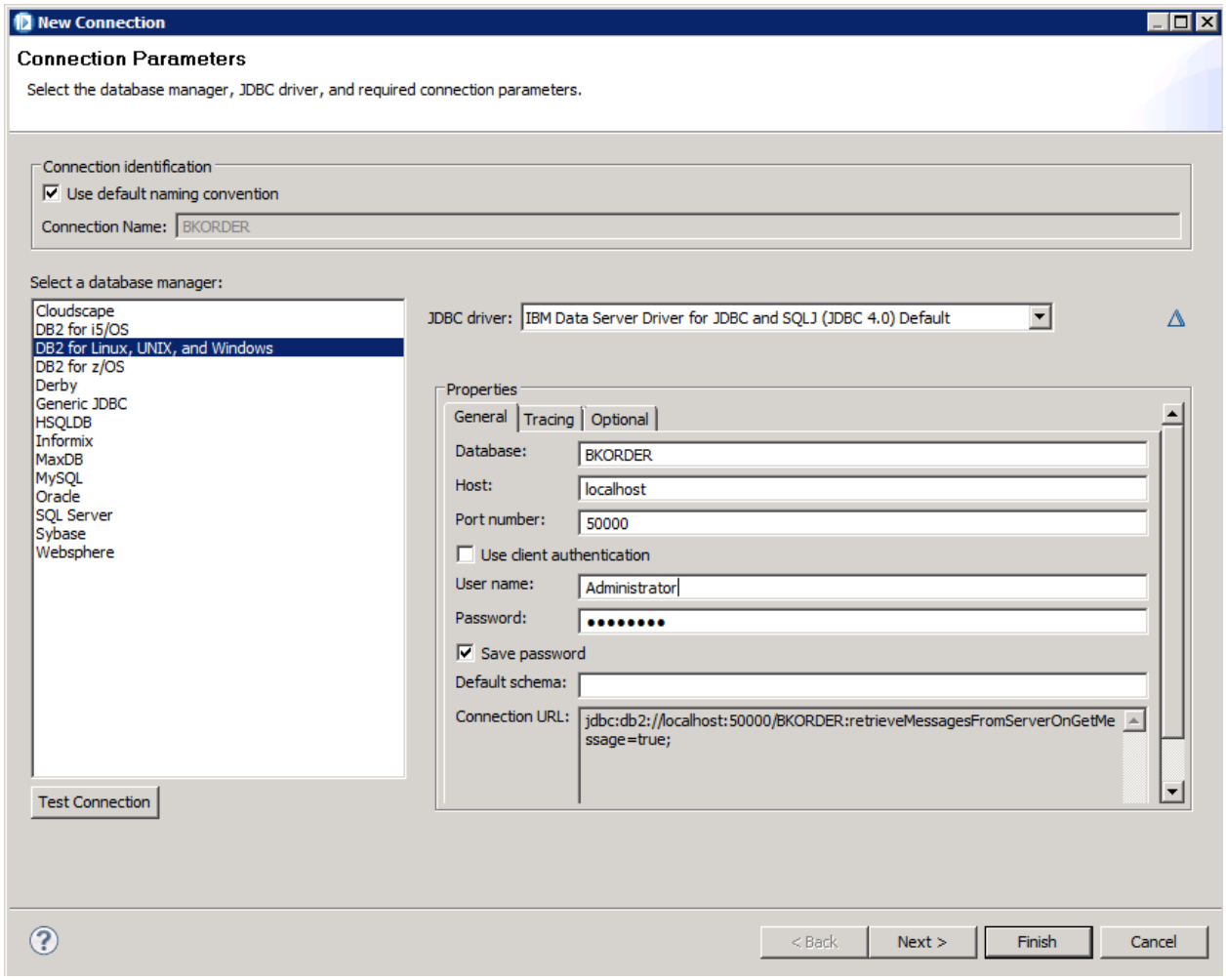
Click import from database to launch the **New database definition file** wizard as shown in figure 4.18.

Figure 4.18 Create a database definition file.



Click **New…** to define a new Data Design project and then select the type of database that you are using. Click **Next >** and set-up the connection parameters to your BKORDER Database as shown in figure 4.19.

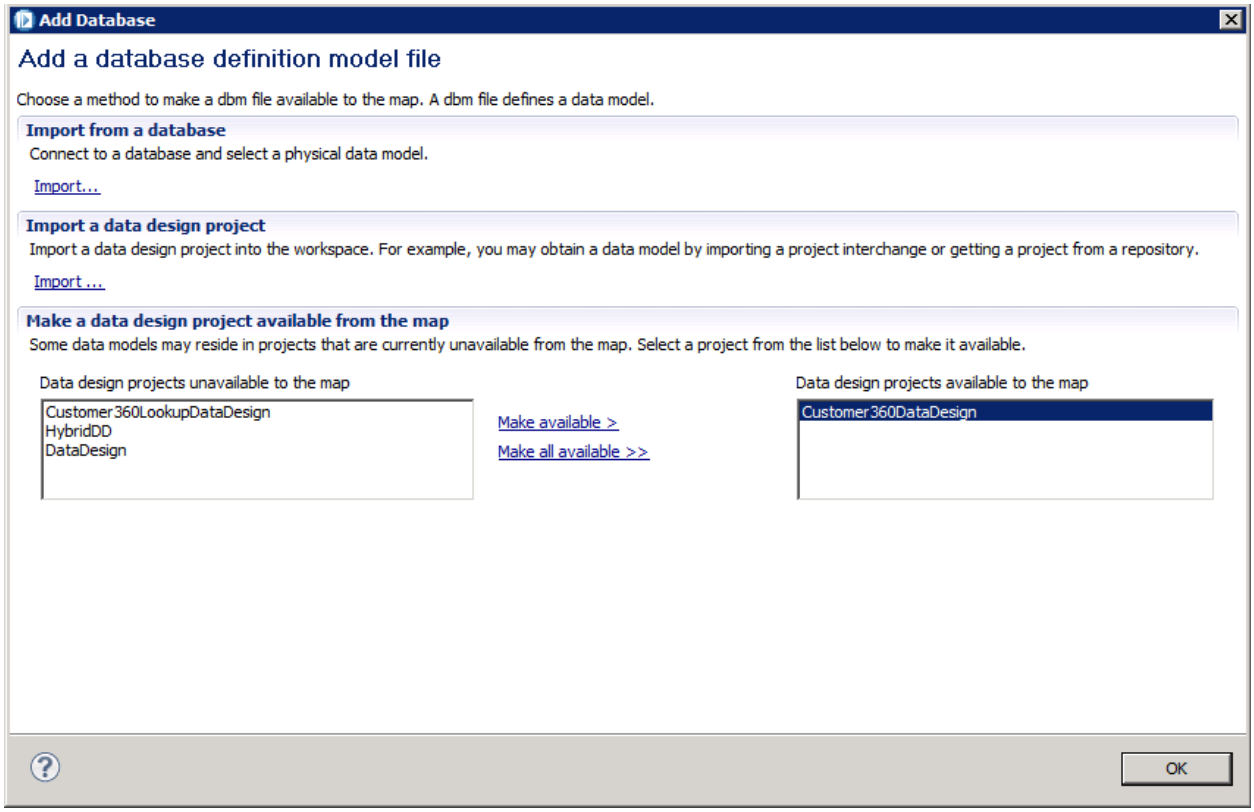
Figure 4.19 Database Connection Parameters



The default naming convention for **Connection Name** is the database name. This name needs to match the JDBCProvider configurable service that is set up in the integration node to connect to the BKORDER database as shown in the Integration Explorer in Figure 4.8. If the configurable service name is different from the default, e.g. BOOKORDER, untick **Use default naming convention** and enter the connection name to match.

You can setup these characteristics to connect to any databases for which the drivers are available. Test the connection and click **Finish** to make the data design project available to the mapper as shown in figure 4.20.

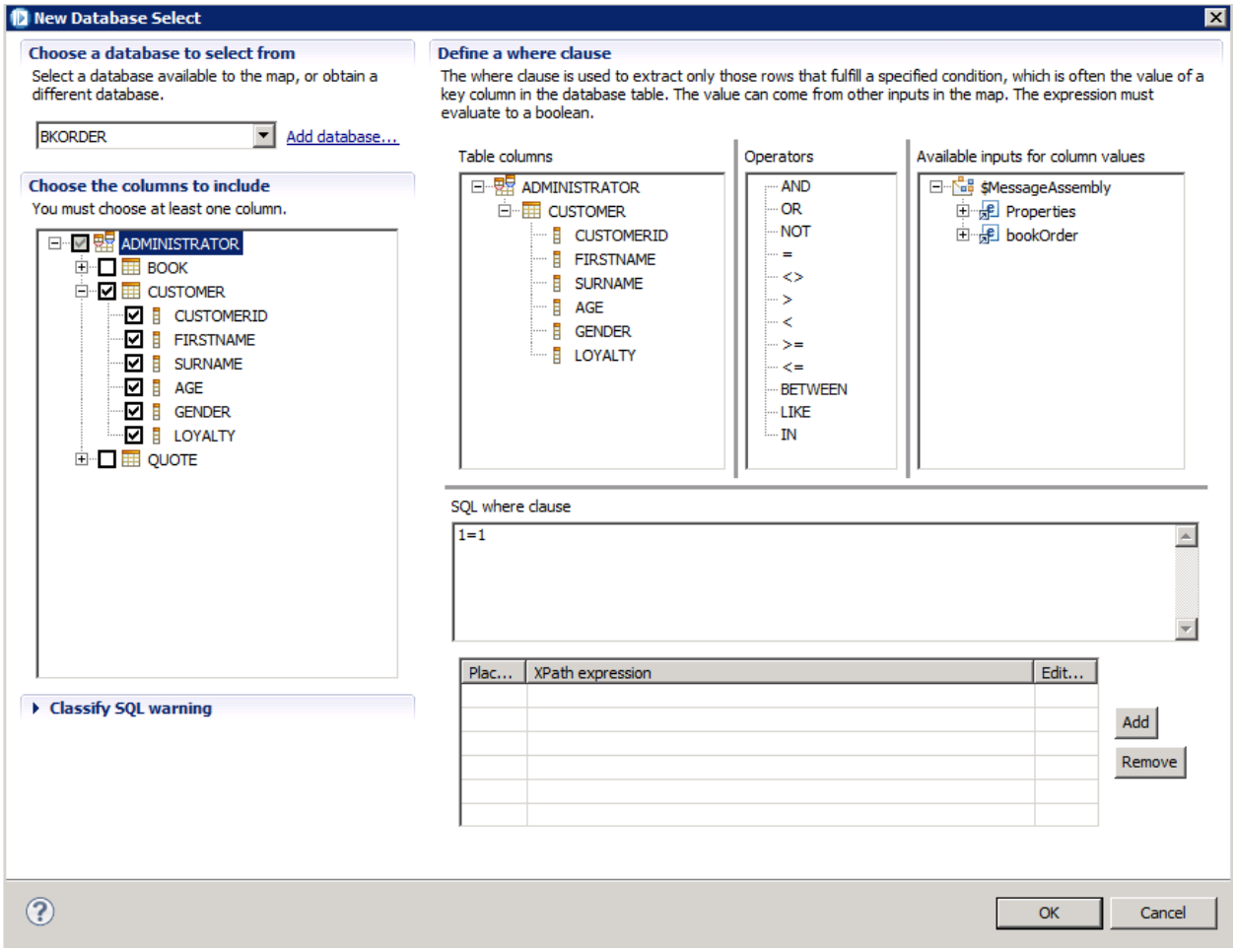
Figure 4.20 Completed database definition model file.



Select the Data design project you created and click OK to close the window and continue. Click **Next >** in the **New Database Definition File** wizard and select the BKORDER connection you created earlier and click **Next >.**
On the next tab select the schema being used – in this case the schema is *Administrator* and click **Finish**.

Click **OK** to close the **Add a database defnition model file** window and bring up the database schema in the mapper **New Database Select** window. In our example we wish to retrieve the *CUSTOMER* table so select this table in the editor as shown in figure 4.21.

Figure 4.21 New database select - table definition.

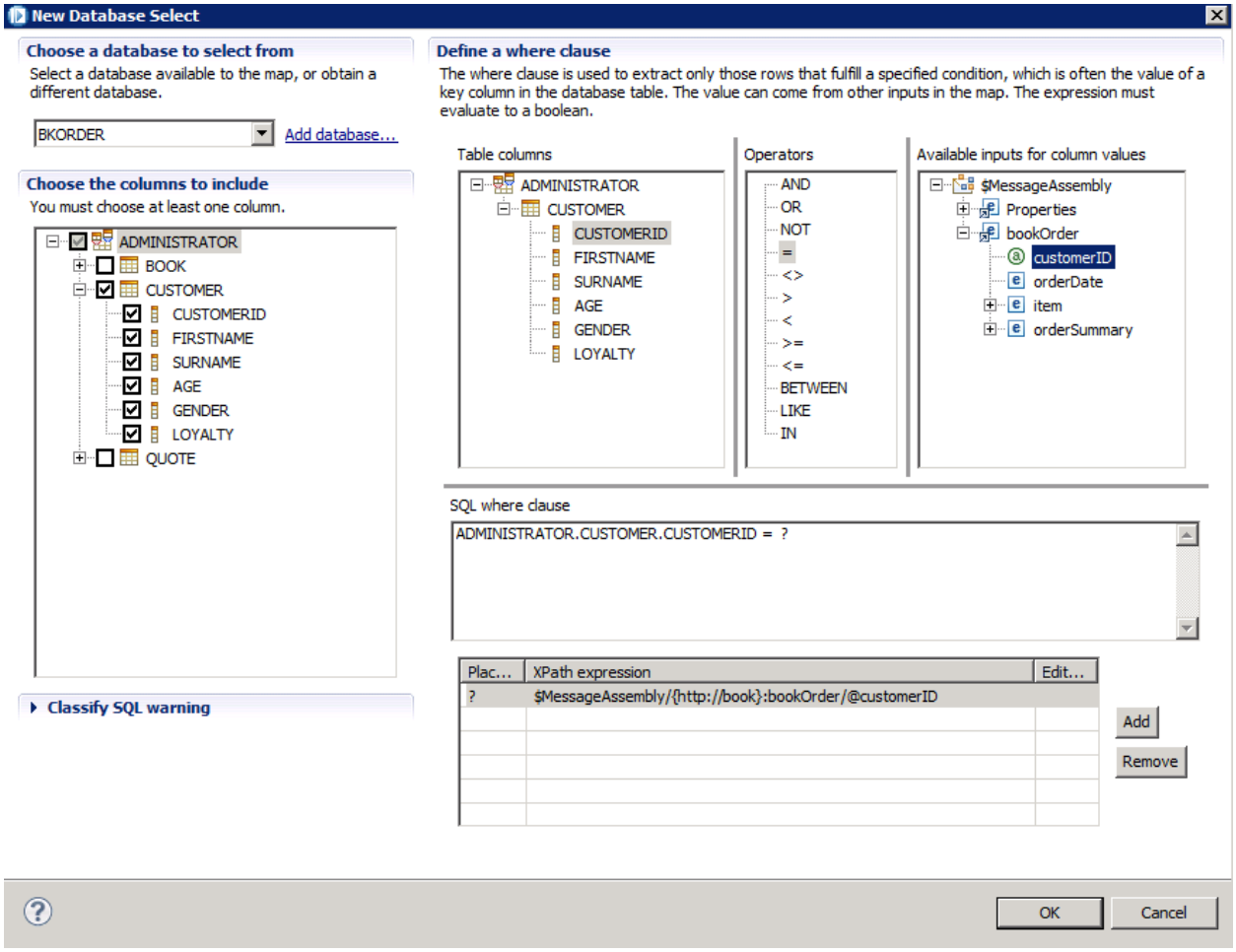


You now need to establish the **SQL where clause**. Delete the default value "*1=1*" and drag the following elements onto the **SQL where clause** panel;

- Table columns: ADMINISTRATOR.CUSTOMER.CUSTOMERID
- Operators: =
- Available inputs for column values: $MessageAssembly/bookOrder/@customerID

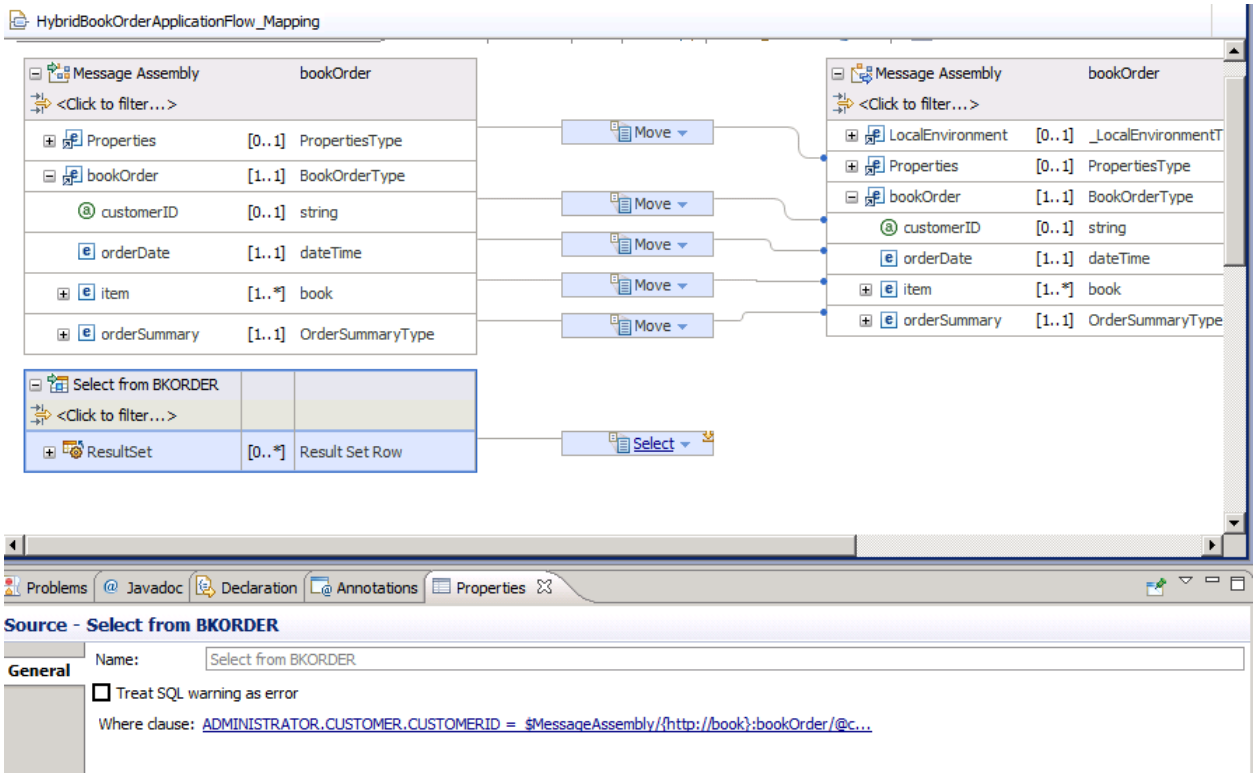
This should result in a panel that looks as shown in figure 4.22.

Figure 4.22 Completed "New Database Select" where clause



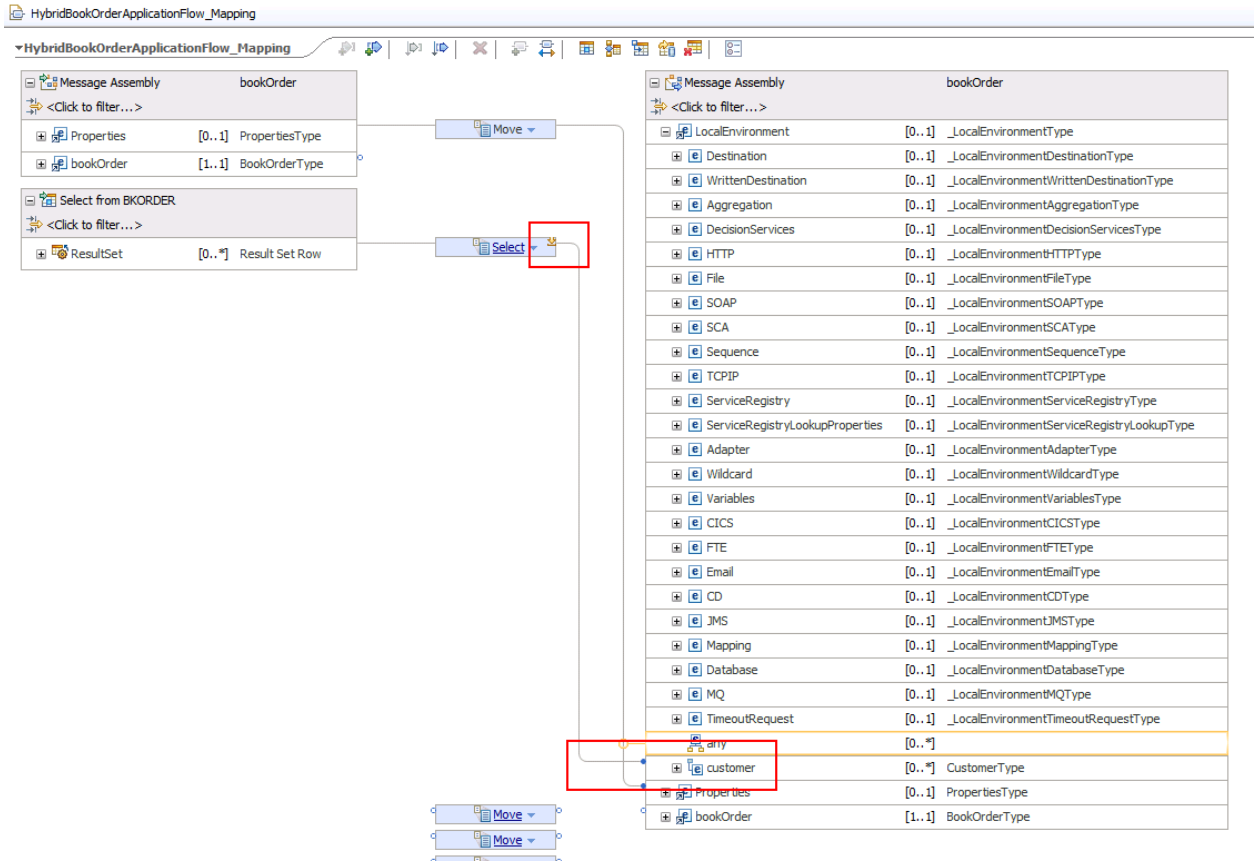
This defines how the mapper defines the query sent to the database. Click **OK** to close the editor and return to the mapping editor as shown in figure 4.23.

Figure 4.23 Mapping editor with completed database select task



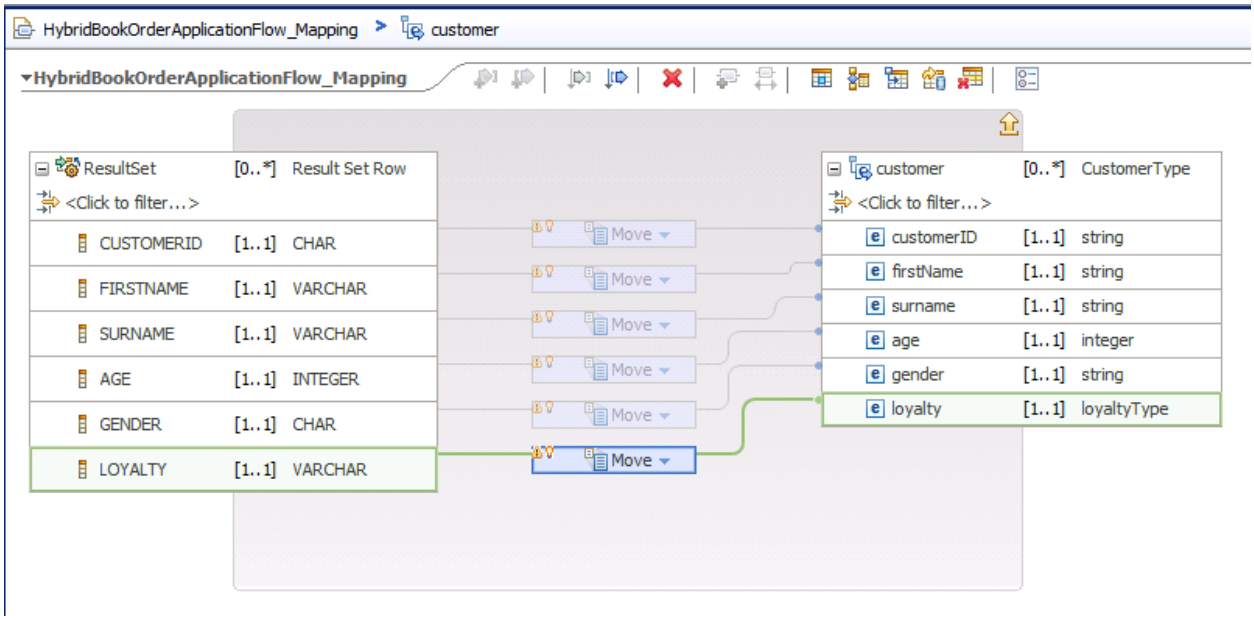
The final step is to map the result of the query to the customer element set-up in the local environment. Expand the **LocalEnvironment** twistie and close the others. This will allow you to drag a connection from the select task to customer as shown in figure 4.24.

Figure 4.24 Mapping the select result set to the message assembly.



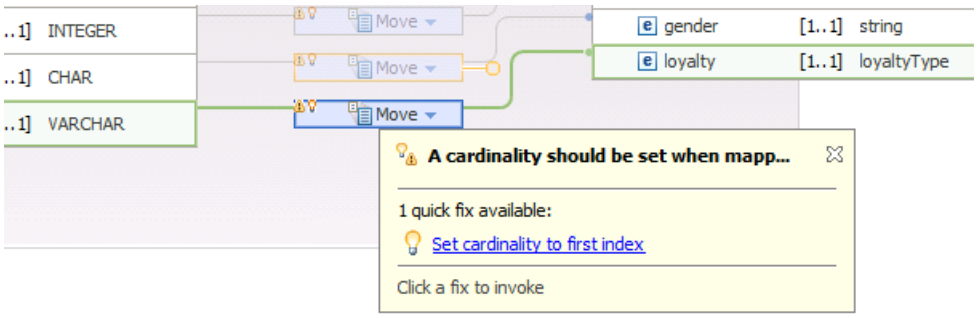
This establishes the relationship between the database result set and the customer element. You now have to map from the table columns to each of the customer fields. **Double click** the **Select** task to bring up the mapping between the table and customer elements. Drag each column across to its equivalent element as shown in figure 4.25.

Figure 4.25 Mapping Table columns to element fields.



The final step is to remove the warnings on the **Move** tasks due to a **Results Set** having potentially multiple rows. Select one of the light bulbs associated with the errors and select the quick fix as shown in figure 4.26.

Figure 4.26 Quick Fix to map cardinality.



Clicking "**Set cardinality to first index**" should resolve the warnings. Save your work and close the mapping editor.

This should now have completed the configuration of the Customer 360 node.

Purchase Scoring Node

The Purchase Scoring Node is implemented in a JavaCompute node that calls the SPSS scoring service using SOAP. It would be possible to use other forms of node or API but this approach illustrates how the SPSS scoring service API needs to be manipulated.

As described in the scoring service Java client in the pattern 3 section, a JAX-WS client (SPSS_Scoring Client.jar) first needs to be generated and added to the JavaCompute node project class path.

In this article this node is implemented in Java and starts with the standard template produced when creating a JavaCompute node in a flow.

Listing 4.1 JavaCompute Node Java template

```
package odm.book;
public class PurchaseScoring extends MbJavaComputeNode {

public void evaluate(MbMessageAssembly inAssembly) throws MbException {
	MbOutputTerminal out = getOutputTerminal("out");
	MbOutputTerminal alt = getOutputTerminal("alternate");
	MbMessage inMessage = inAssembly.getMessage();
	MbMessageAssembly outAssembly = null;
	try {
		// create new message as a copy of the input
		MbMessage outMessage = new MbMessage(inMessage);
		outAssembly = new MbMessageAssembly(inAssembly, outMessage);
		// ----------------------------------------------------------
		// User code
		// ----------------------------------------------------------
	} catch (MbException e) {
		// Re-throw to allow Broker handling of MbException
		throw e;
	} catch (RuntimeException e) {
		// Re-throw to allow Broker handling of RuntimeException
		throw e;
	} catch (Exception e) {
		// Consider replacing Exception with type(s) thrown by user code
		// Example handling ensures all exceptions are re-thrown to be
		// handled in the flow
		throw new MbUserException(this, "evaluate()", "", "", e.toString(),
					null);
	}
	// The following should only be changed
	// if not propagating message to the 'out' terminal
	out.propagate(outAssembly);
```



```
}}
```

The user code is then replaced with the code shown in listing 4.2 to manipulate the message tree and environment.

Listing 4.2 Message tree manipulation

```
MbElement outRoot = outMessage.getRootElement();
MbElement envtRoot =
      outAssembly.getLocalEnvironment().getRootElement();
// Retrieve the customer from the environment
MbElement customer =
      envtRoot.getFirstElementByPath("customer");
//create the score element entry
MbElement score =
      envtRoot.createElementAsLastChild(MbElement.TYPE_NAME);
score.setName("score");
score.setNamespace(customer.getNamespace());
//Now do the call to SPSS to get the score
getScore(score, customer);
```

Note how the *customer* element is retrieved from the local environment where it was inserted by the customer 360 mapper node.

The getScore() operation then uses the information in the *customer* element and the returned score to populate the *score* element.

Listing 4.3 getScore() operation

```
private void getScore(MbElement score, MbElement customer) {
try {
      //Retrieve the customer gender and loyalty from the customer     String
gender =
        customer.getFirstElementByPath("gender").getValueAsString();
      String loyalty =
        customer.getFirstElementByPath("loyalty").getValueAsString();
      //Add to the score element          score.createElementAsLastChild(MbEle-
ment.TYPE_NAME,
            "loyalty",loyalty);
      score.createElementAsLastChild(MbElement.TYPE_NAME,
            "gender",gender);
      //Create the client and connection
      OrderPlacedScoringClient client = new OrderPlacedScoringClient();
      //make the score call
      client.getScore(loyalty, gender);
      //Populate the response            score.createElementAsLastChild(MbEle-
ment.TYPE_NAME,
            "orderPlaced",client.getPredictedOrderPlaced());
      score.createElementAsLastChild(MbElement.TYPE_NAME,
            "confidence",client.getPredictedConfidence());
} catch (MbException e) {
      e.printStackTrace();
}
```



```
}
```

For details of the client see the listings in pattern 3.

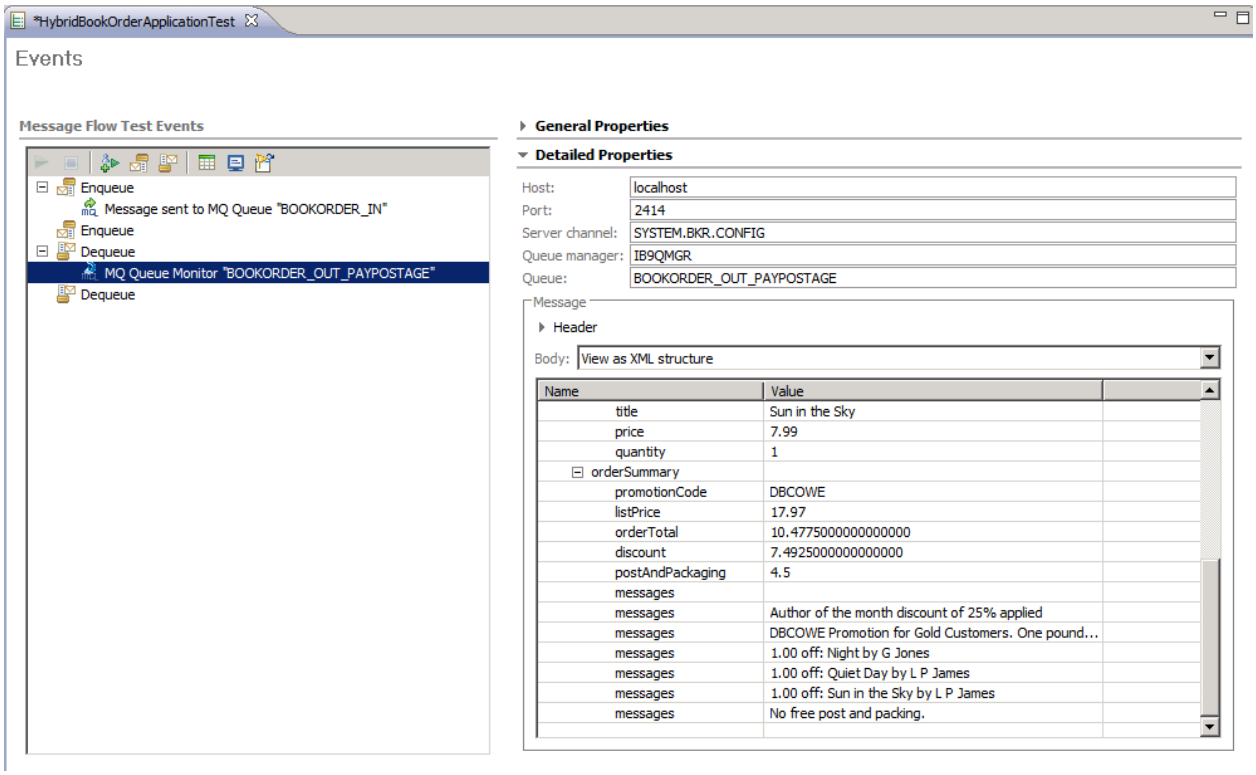
Pattern Testing

The simplest form of testing when using IBM Integration Bus is to use a Message Flow Test as show in figure 4.8. The flow test allows messages to be defined and sent to the queues on which the flow is listening. Messages can then be retrieved from the output queues allowing the end to end processing of the flow to be observed (and debugged) within the tooling environment. Two test messages have been established.

The first book order message is for a Gold male customer, ID = F-666. The results of this request are shown below in figure 4.27.

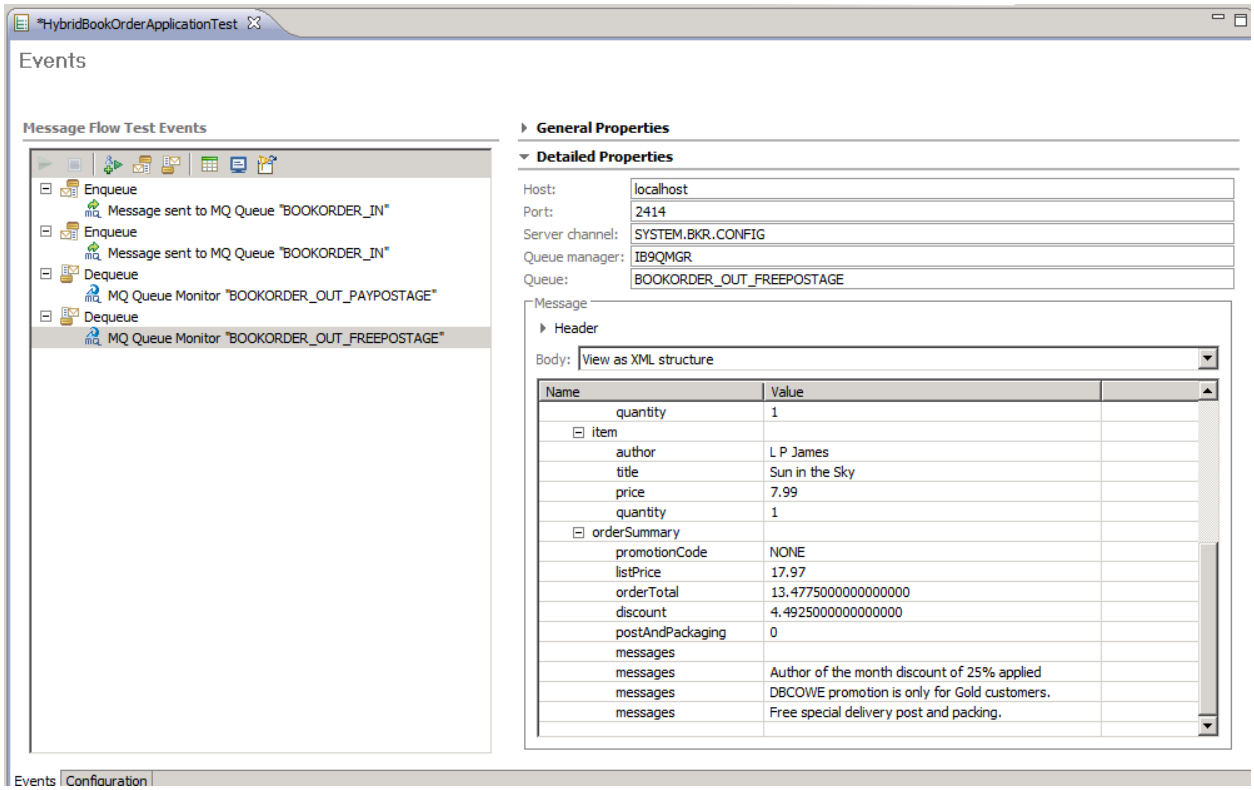
The second case is for a Bronze male customer, ID=B-222. The results of this request are shown in figures 4.28.

Figure 4.27 Gold Male customer order



This order has a DBCOWE promotion code selected which will be applied as the customer has a GOLD loyalty retrieved from the customer 360 degree lookup. Gold male customers are likely to purchase a quote according to the scoring and are therefore not offered free postage.

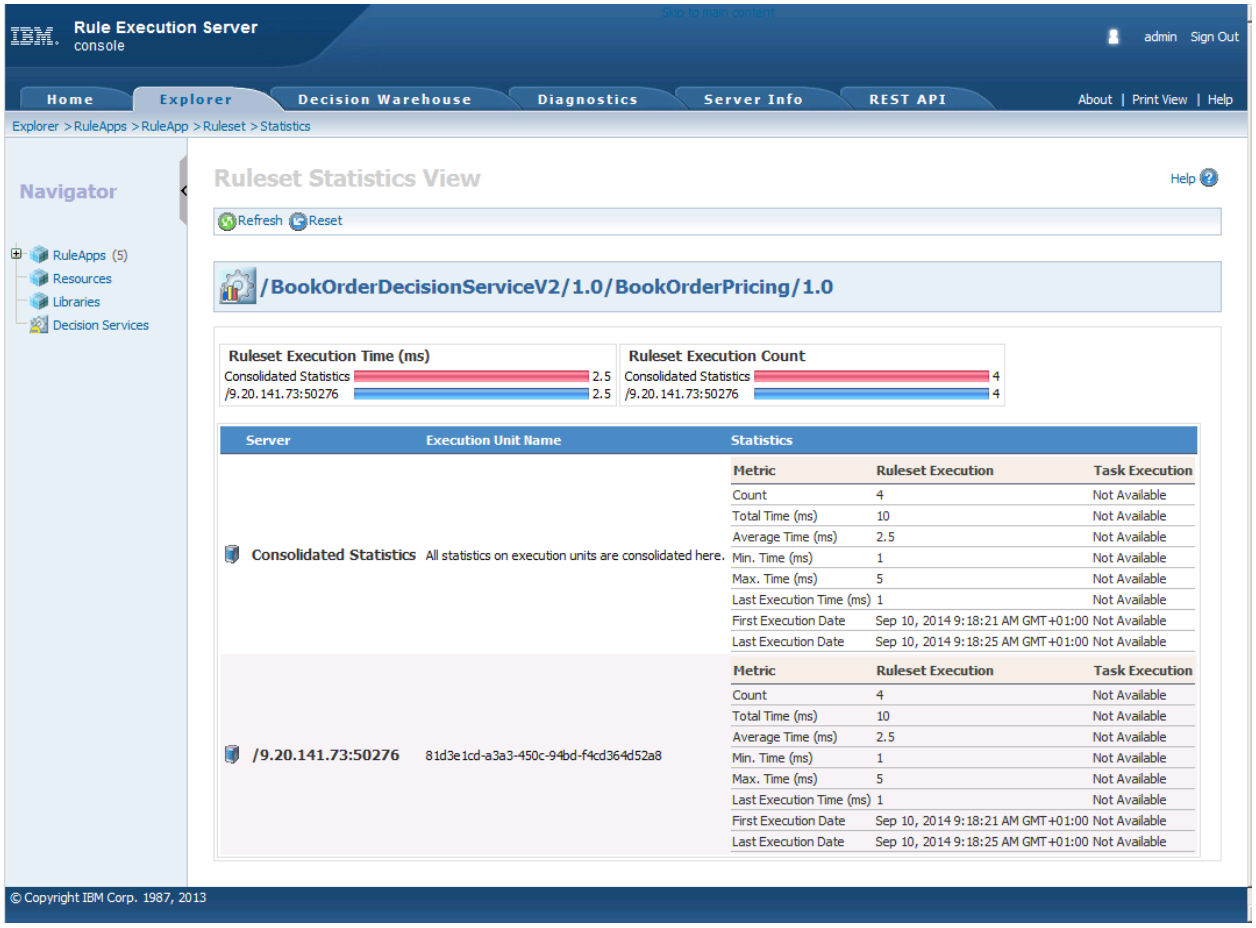
Figure 4.28 Bronze Male customer order



In this case as the customer is ranked as BRONZE, they are not eligible for the DBCOWE promotion. As they are likely to not place an order, they are eligible for free post and packing. As the confidence that they would reject the quote was high, they get the "Free special delivery post and packing". In this case the message has also been routed to the BOOKORDER_OUT_FREEPOSTAGE queue.

If the configurable decision service repository has been enabled for trace, the execution statistics are also available in the Rule Execution Server console as shown in figure 4.29.

Figure 4.29 RES console Ruleset statistics

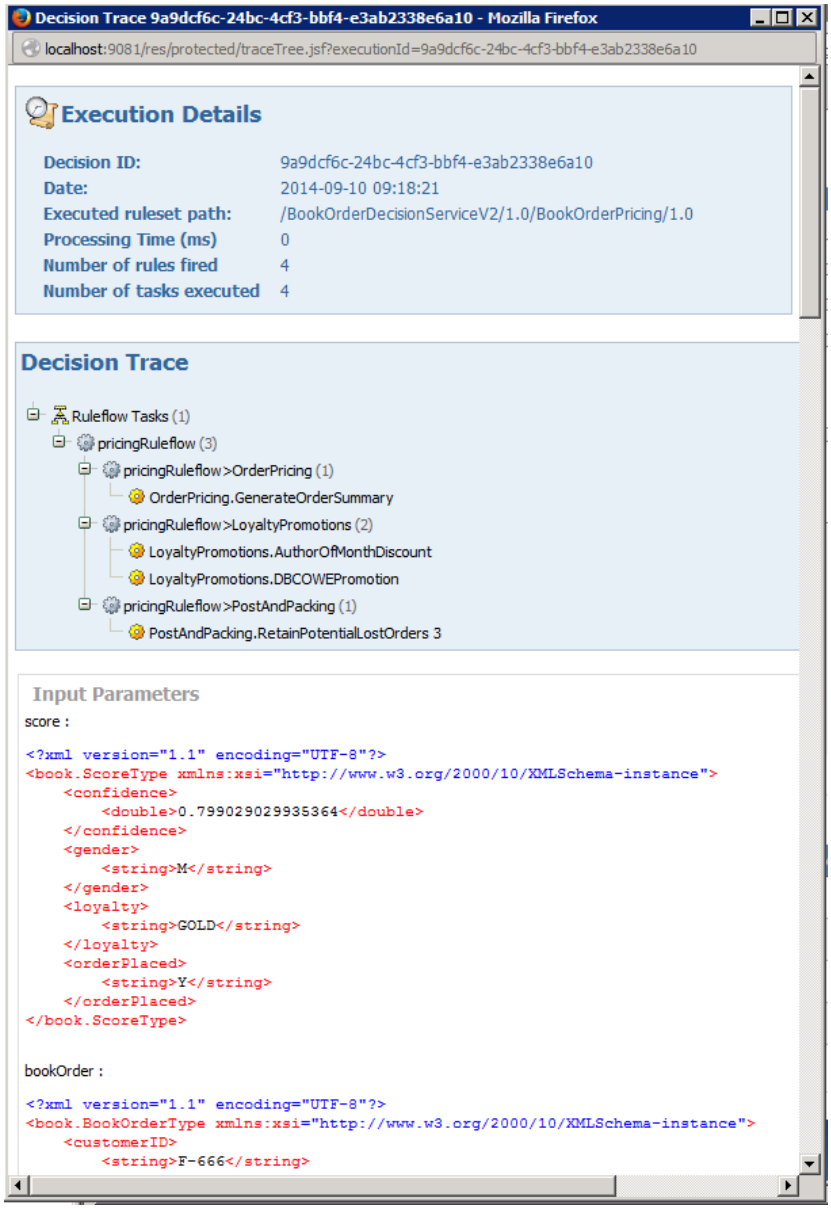


The Ruleset Statistics are useful to determine if the rules are being executed from the IIB flow and approximate execution times.

Ruleset execution times can vary with a ruleset that has to be loaded into memory (on first ruleset execution) taking longer than subsequent executions. It is slower than might be expected as the decision warehouse has also been enabled in this scenario.

It can also be useful when debugging rulesets to enable monitoring in the decision warehouse. Each decision is recorded in the decision warehouse and searches can be made to find the decisions of interest. Each decision report shows the input and output parameters together with the rules executed as shown in figures 4.30 and 4.31.

Figure 4.30 Decision Warehouse Trace – Gold customer

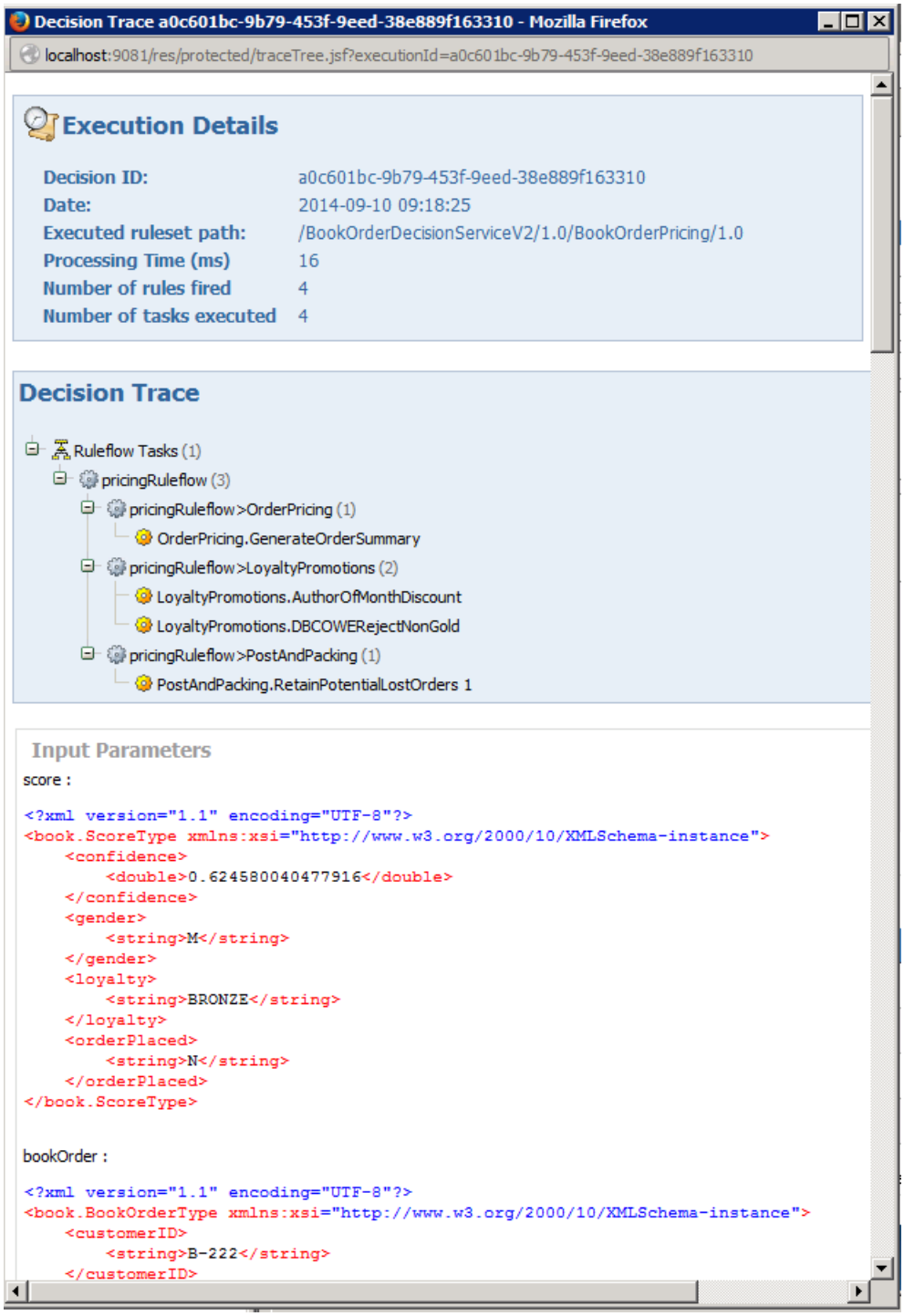


In this decision warehouse trace you can see the score input parameter containing the customer loyalty (GOLD) and gender (M) retrieved from the Customer 360 node for customer F-666 and the orderPlaced (Y) and confidence (0.799) returned from the scoring service.

At the top of the panel you can see the rules that fired showing the AuthorOfMonthDiscount, DBCOWE promotion and the row number (3) that fired in the RetainPotentialLostOrders decision table.

Further down the panel (not shown) you can see the output parameters and any execution output provided (for example from print statements in the rules).

Figure 4.31 Decision Warehouse Trace – Bronze customer



Looking at a trace for a BRONZE, Male customer (B-222) shows that the customer is likely to reject the offer with a confidence of 0.624.
In this case the AuthorOfMonthDiscount rule is applied and the DBCOWERejectNotGold rule means that no DBCOWE discount is applicable.
Row 1 of the RetainPotentialLostOrders decision table then fires allocating the free post and packing.

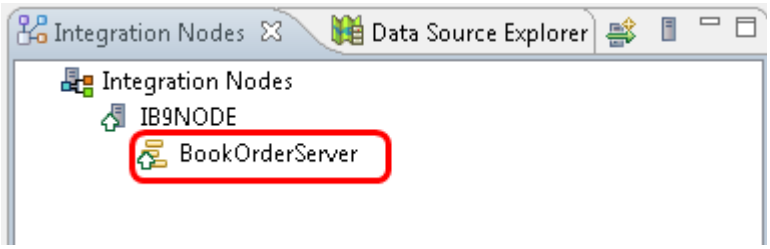
Visual Message Flow Debugging in the Integration Toolkit

The message flow debugger is a visual interface that supports the debugging of message flows running in the integration server from the Integration Toolkit. The message flow debugger is based on the Java debugger provided by the Java development tools. It can also be used to debug JavaCompute and ESQL nodes.

For full details of all the debugging options provided by IBM Integration Bus, see Testing and debugging message flows. This includes the use of traces and the test client as well as the visual debugger.

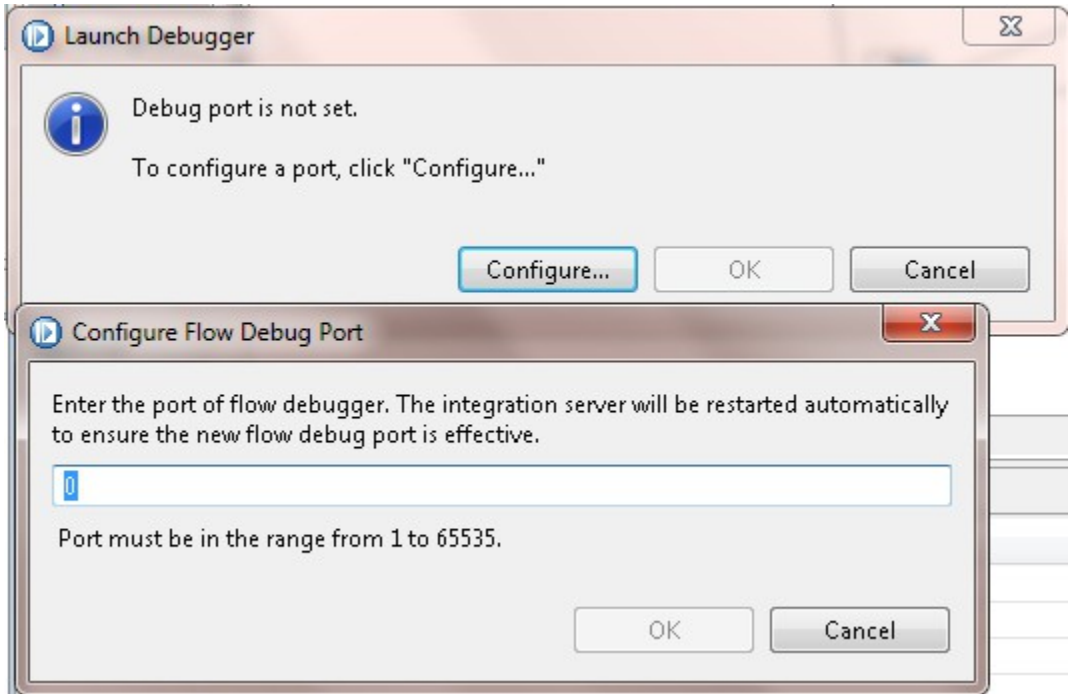
To launch the flow debugger, right-click on the Integration Server in the Integration Nodes view and select **Launch Debugger (Port is <port number>)**.

Figure 4.32 Integration server for launching debugger



If this is the first time the debugger is launched, the port number is shown as 0. The port number can be configured or changed if already configured during the launch.

Figure 4.33 Set debug port for flow debugger



Type in the port number to use and click OK. The Progress view will show the debugger launching to connect to the JVM of the integration server. Once the debugger is launched successfully, the following message pop-up is displayed:

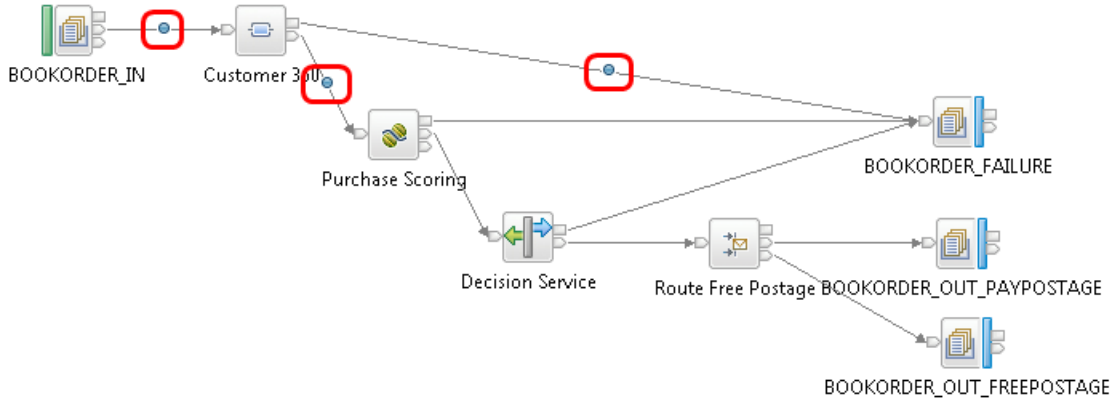
Debugger is launched. Start debugging by adding a breakpoint to the message flow and sending a test message.

You can terminate the debugger any time by right-click on the integration server and select **Terminate Debugger (Port is <port number>)**. It is important to terminate the debugger after debugging session as there is performance impact leaving the debugger launched.

You can add a breakpoint to the message flow by right-clicking on a flow connection arrow of interest and select **Add Breakpoint**. If a node has multiple connections before and/or after the node and you want to add breakpoints on all of these connections, you can right-click on the node and select **Add Breakpoints Before Node** and/or **Add Breakpoints After Node**. You can also right-click on individual breakpoint and select **Remove Breakpoint**.

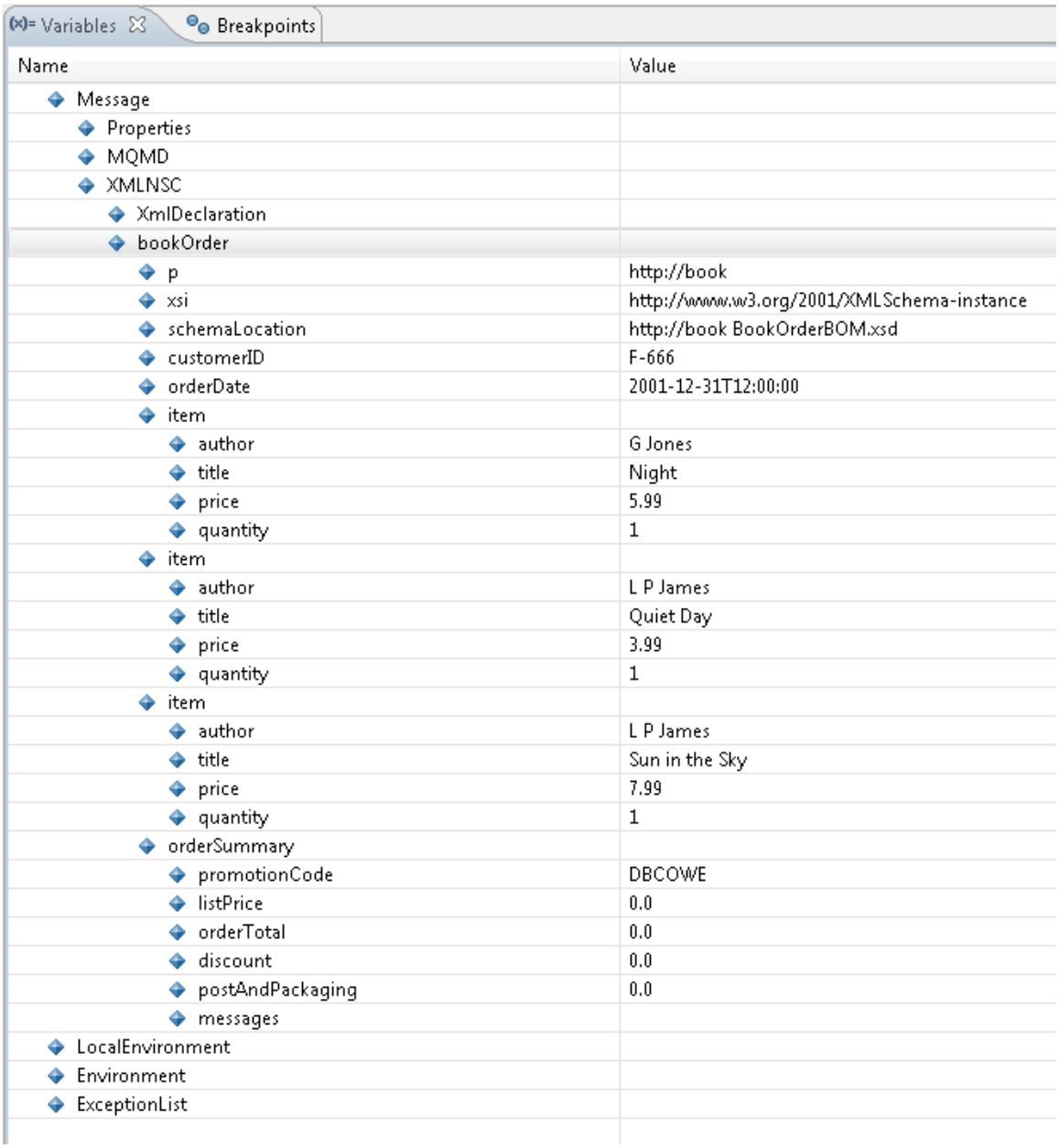
As an example, breakpoints are added before and after the Customer 360 node to examine the message contents before and after the data mapping.

Figure 4.34 Set breakpoints for flow debugger



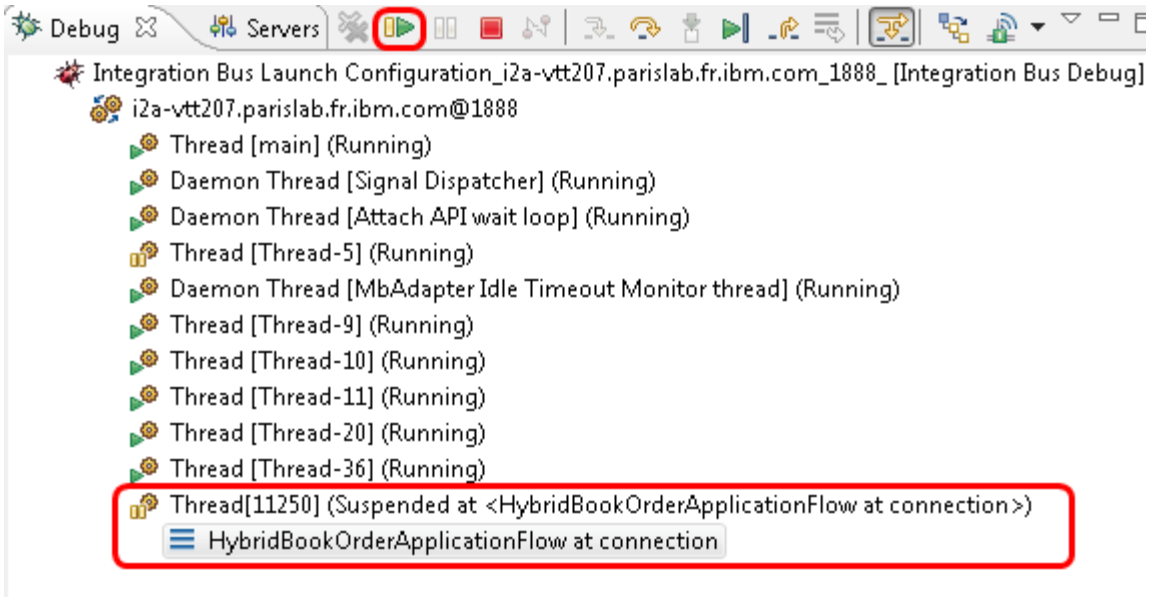
With the debugger launched, sending a test message will get a prompt to switch to debug perspective. The debug view for the Gold Male customer order shows the following message content at the first break point in the Variables panel. This is the message received from the MQInput node BOOKORDER_IN showing the order details. At this stage, there is no data in LocalEnvironment, Environment and ExceptionList. Apart from examining the data, you can also change the values of data for debugging purpose.

Figure 4.35 Message from BOOKORDER_IN node



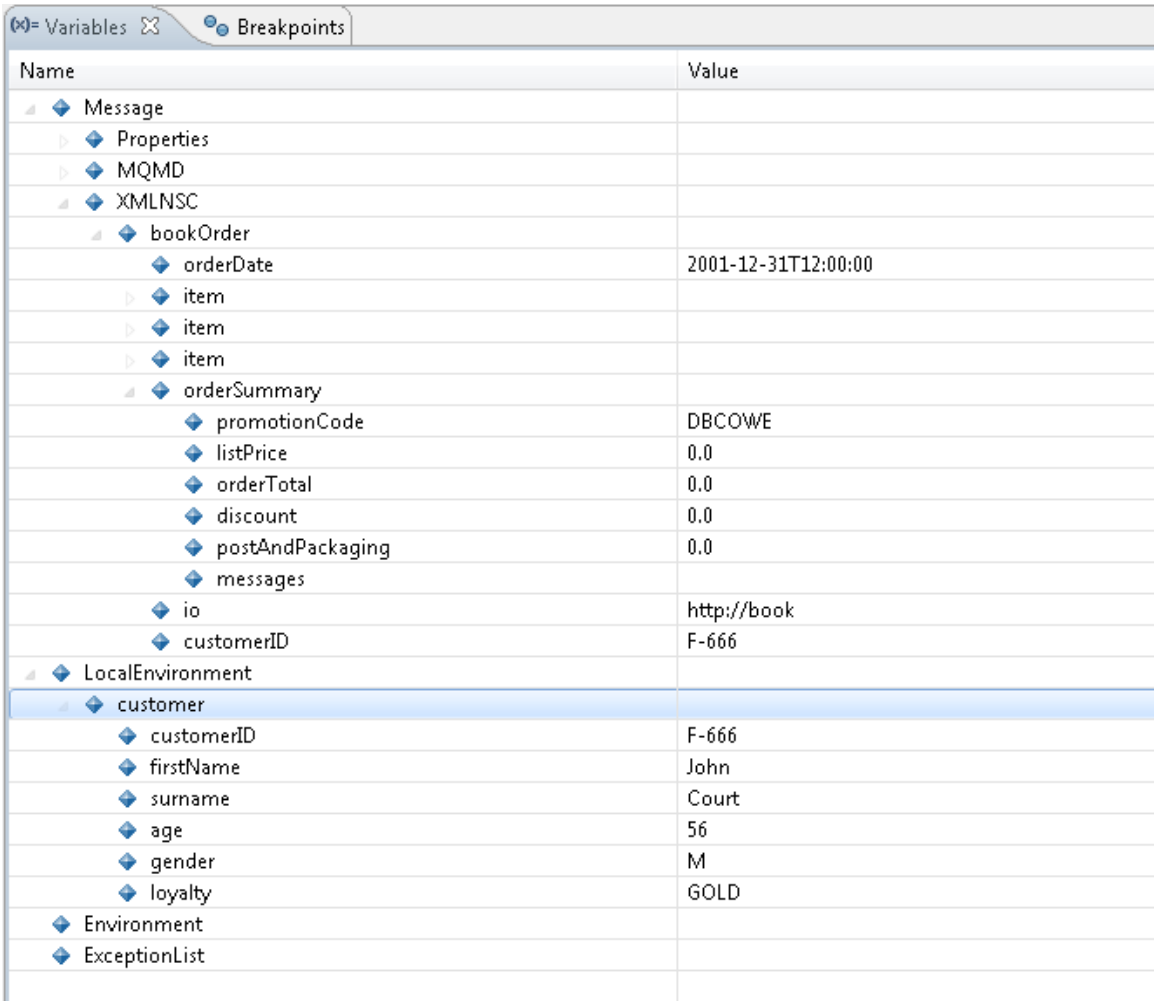
To resume and continue to the next break point which is set after the Customer 360 node, click on the Resume icon to resume running the suspended thread.

Figure 4.36 Resume debugging



The suspended thread goes into the Running state and is then suspended at the next break point after the Customer 360 node.

Figure 4.37 Message from Customer 360 node



The Customer 360 node retrieves the customer details from the BKORDER database based on the CustomerID and the customer data is added to the LocalEnvironment as seen in the Variable panel.

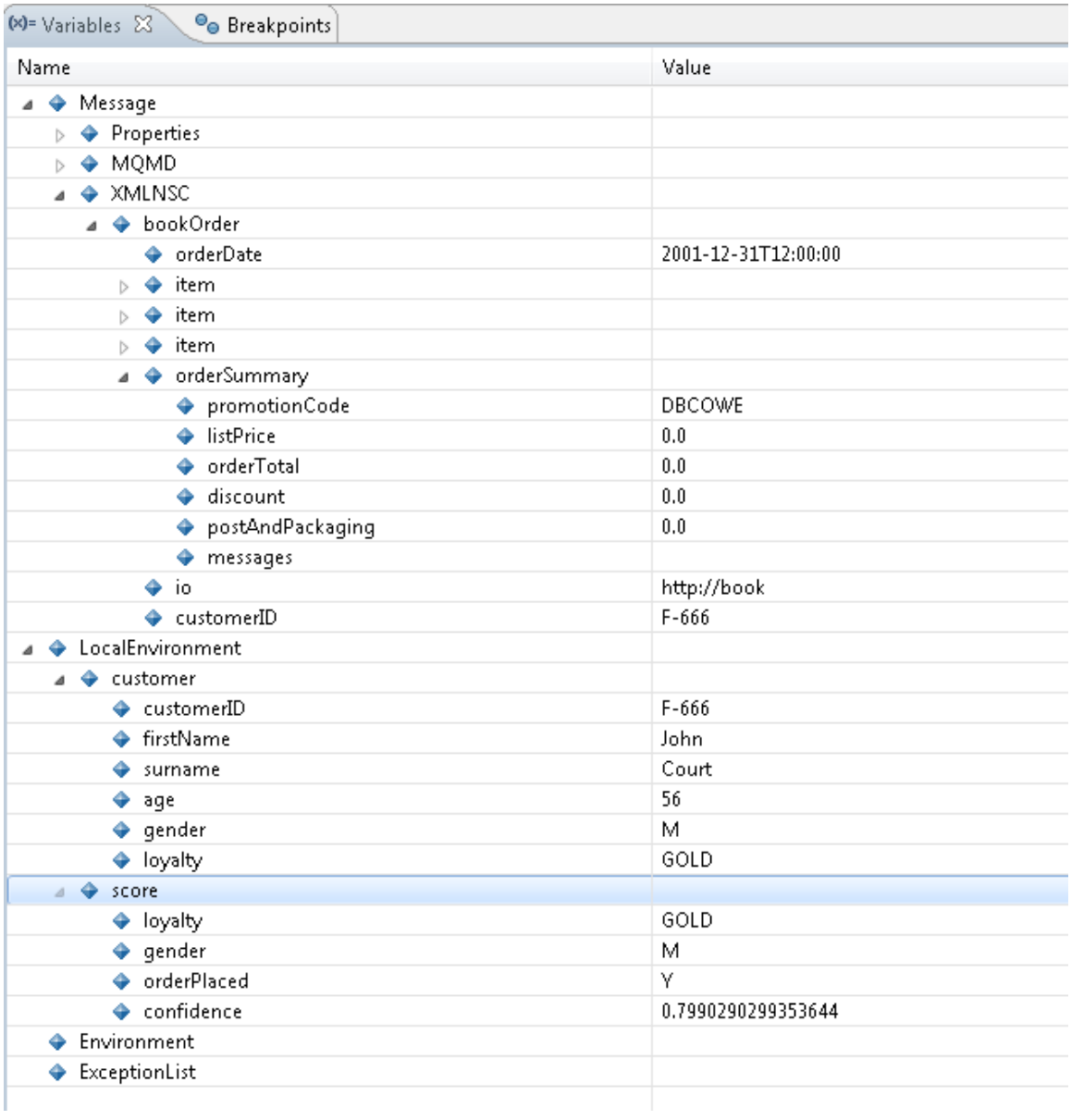
Comparing the bookOrder data before and after the node shows some differences in the bookOrder details due to the way bookOrder is mapped as described in Customer 360 Node. By mapping sub elements of bookOrder rather than the bookOrder itself, the Mapping node builds and populate the output data explicitly and is driven based on the XML schema model of the target. This mapping, which does not change the contents of the order, allows the bookOrder to be serialized correctly when the message arrives at the DecisionService node later.

As this is the last break point set in the message flow, the suspended thread will run to the end when the **Resume** button is clicked. To continue debugging, additional breakpoints can be added to the message flow. Alternatively, you can also click on the **Step Over** button to step

over and suspend after the next node in the flow. There is also an **Enable Step-by-Step Mode** toggle button  which, if enabled, suspends after next node when **Resume** button is clicked.

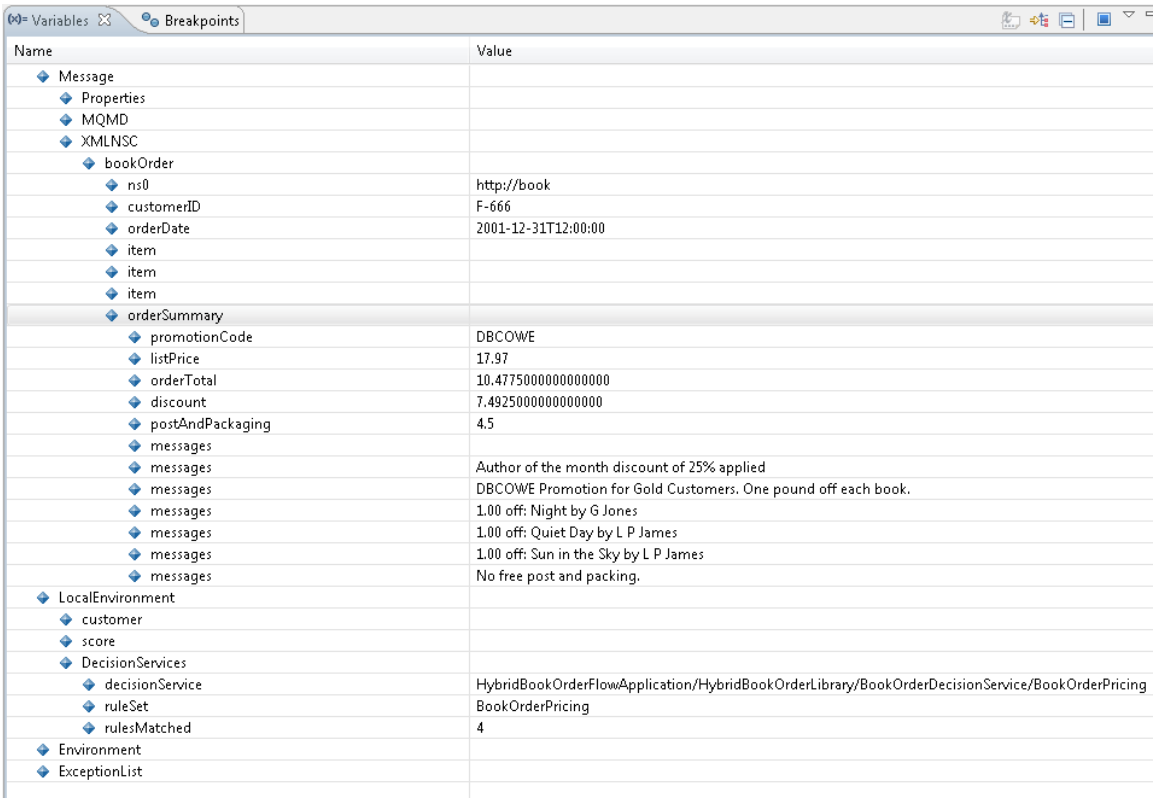
Continuing debugging, the message structure can be examined after each node transformation. Suspending the flow after the Purchase Scoring node shows that the score data has been added to the LocalEnvironment.

Figure 4.38 Message from Purchase Scoring node



The next "step over" provides the results of the pricing decision in the Decision Service node.

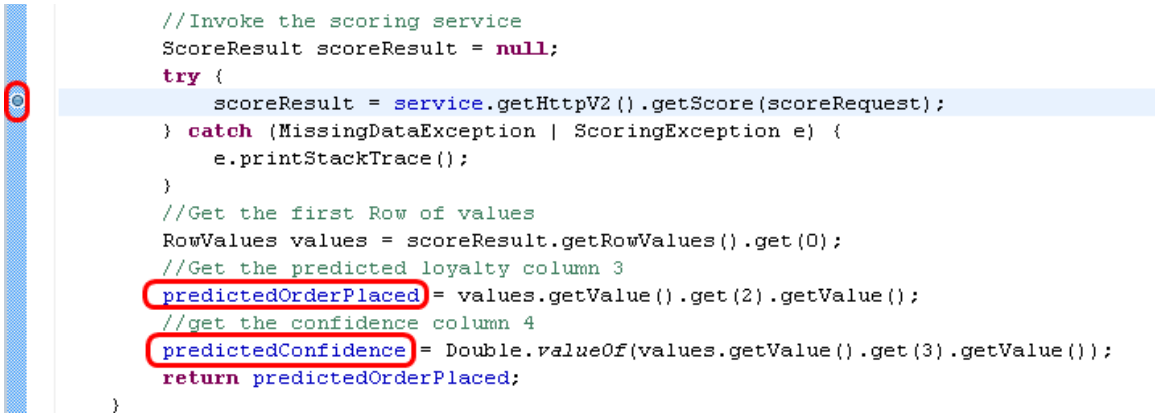
Figure 4.39 Message from Decision Service node



The bookOrder data which is to be included in the response message shows that the orderSummary is updated by the Decision Service node from the Ruleset execution based on the customer and score data in the LocalEnvironment. The LocalEnvironment also shows data from the Decision Service node with details of the decision service and ruleset executed and the number of rules matched.

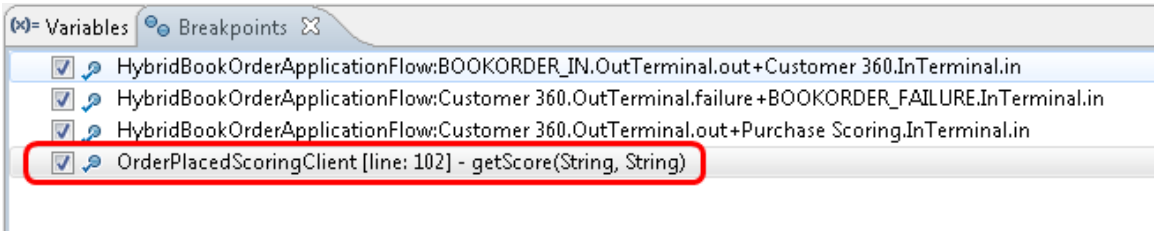
The message flow debugger can also debug JavaCompute nodes. In the Java class editor, right-click on the left-most column (shown blue in Figure 4.40 below) for the line of code you want to set the breakpoint and selects **Toggle Breakpoint**. In this example, the breakpoint is set before the getScore Java method is called to trace the predictedOrderPlaced and predictedConfidence values.

Figure 4.40 Setting a Java breakpoint



The Java breakpoint can be seen listed with the previously set message flow breakpoints in the debug view.

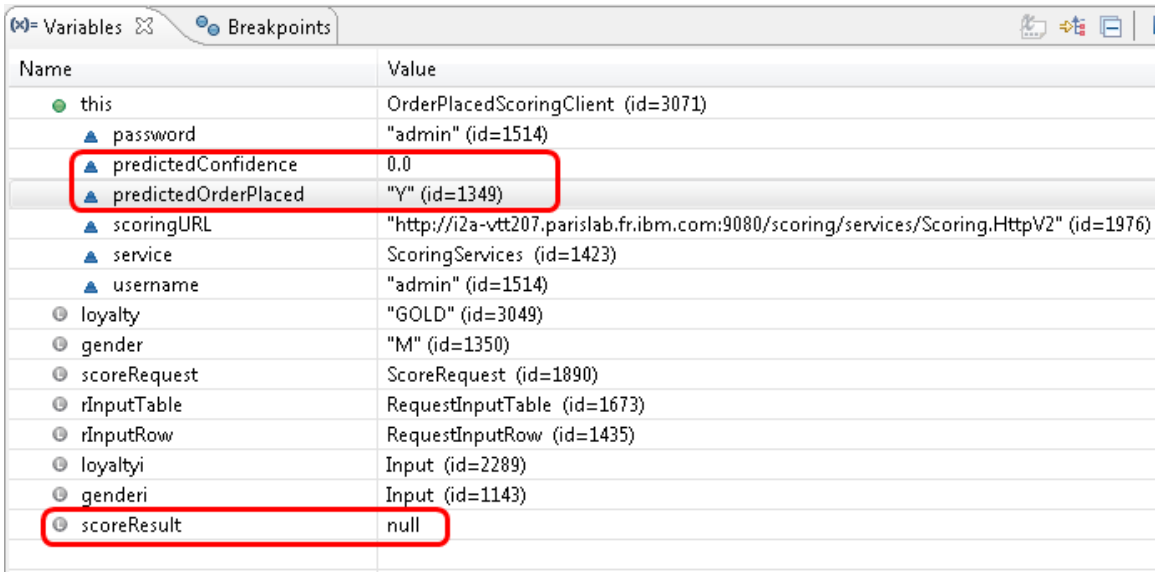
Figure 4.41 Java breakpoint listings



When the thread is suspended at the OrderPlacedScoringClient breakpoint, the Java debug view shows scoreResult as null and predictedConfidence and predictedOrderPlaced values with the initialised values of 0.0 and Y respectively before calling the SPSS scoring service.

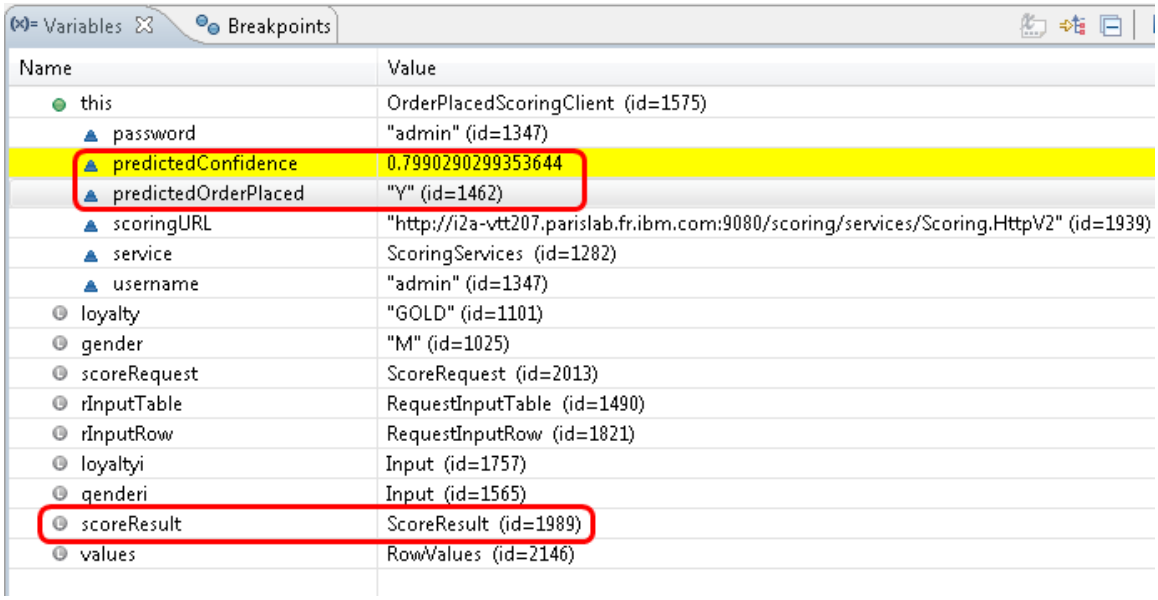
For the first run, you need to click on the **Edit Source Lookup Path…** button in the Java editor for the Java class and select the Java project in the workspace to view the Java source code in the debugger view without switching to the Integration Development view.

Figure 4.42 Java variables before calling SPSS scoring service



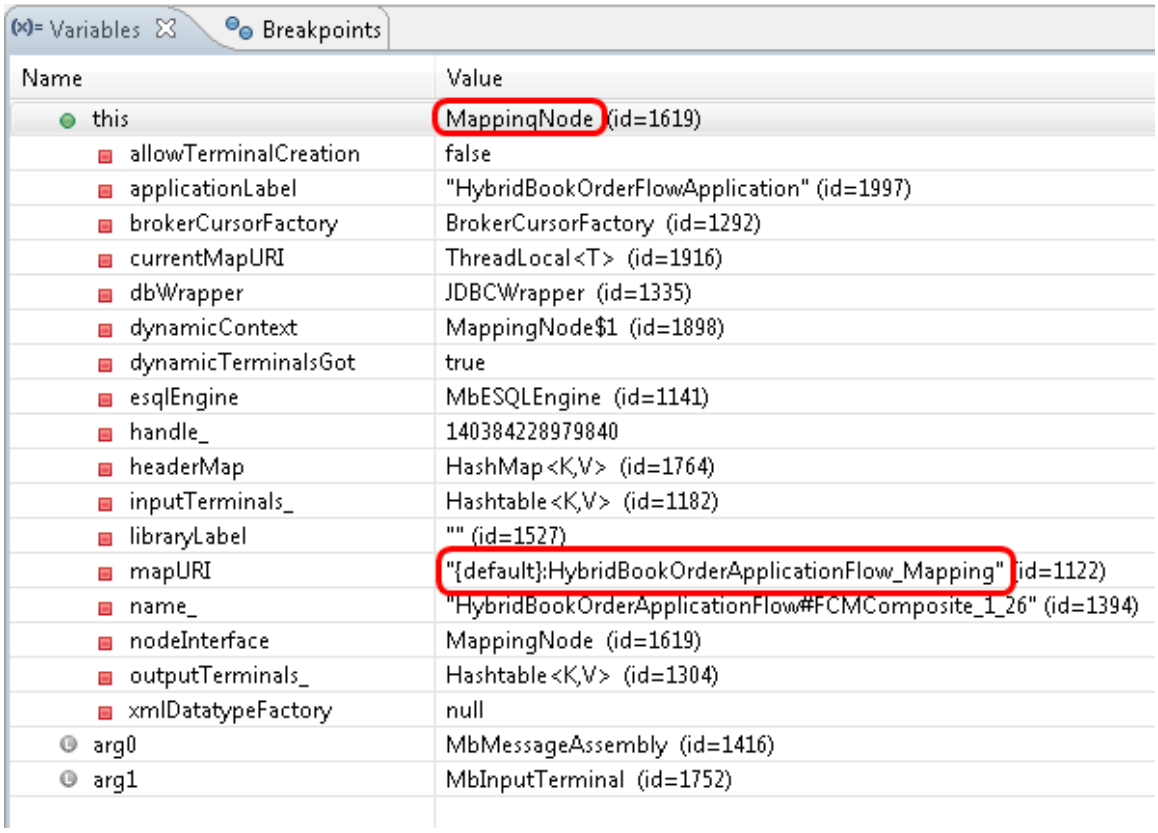
Stepping over the code in the OrderPlacedScoringClient Java class and examining the variables: predictedConfidence and predictedOrderPlaced shows how they are updated with results from the SPSS scoring service by the getScore method.

Figure 4.43 Java variables for scoring data



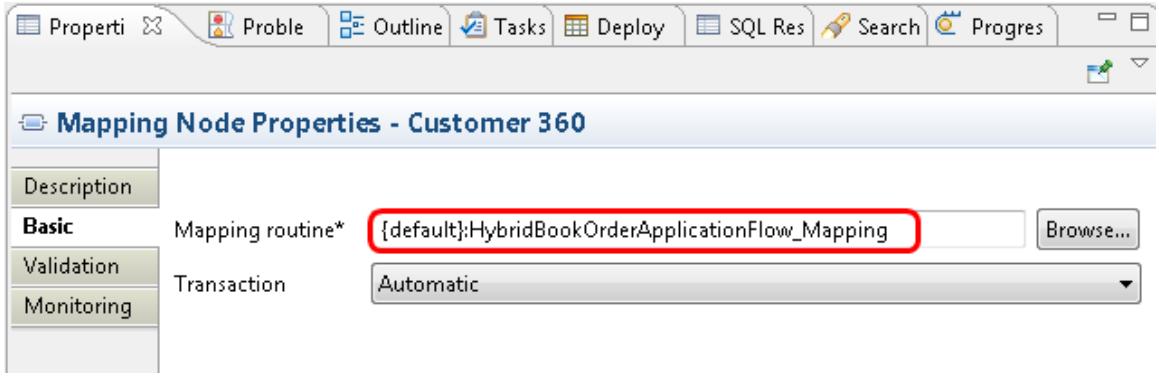
The message data has an exceptionList which is useful for debugging problems in the message flow. As an example, we show a problem in the Customer 360 node when the BKORDER configurable service is not set up in the server environment. In this case, the running thread suspends when the node runs into an Exception as shown in figure 4.44.

Figure 4.44 Suspend in node with Exception



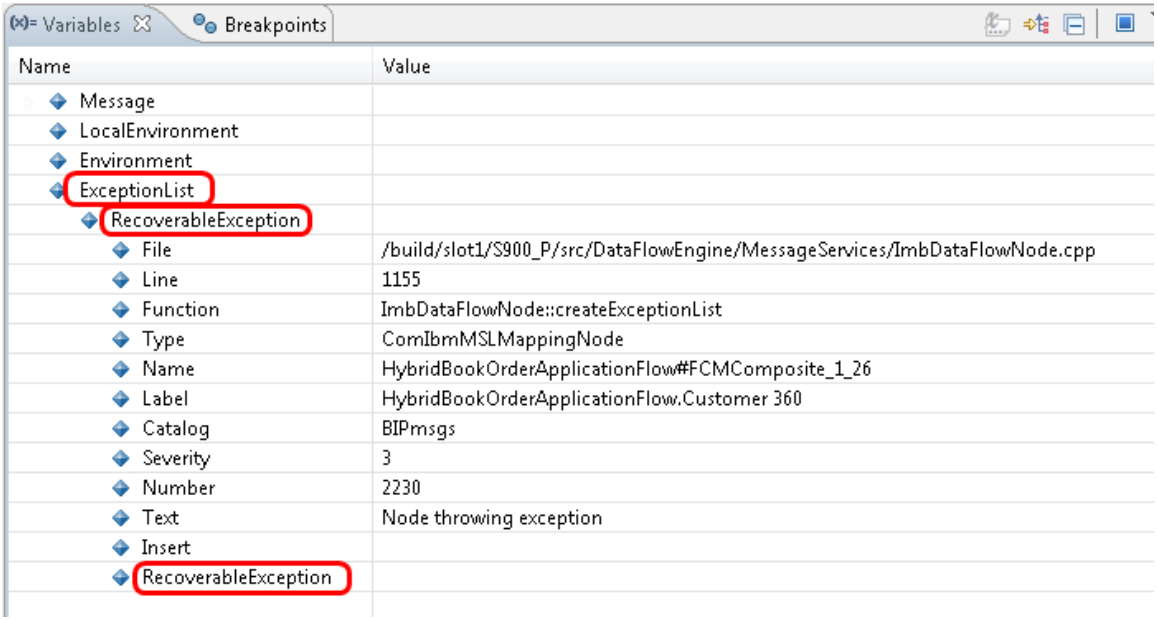
From the Variables, we can see this is a MappingNode with mapURI as {default}:HybridBookOrderApplicationFlow_Mapping. The mapURI is the mapping routine in MappingNode as shown in the basic properties for Customer 360 node in figure 4.45. Other node types will have different properties to identify the node in the message flow.

Figure 4.45 Mapping node properties



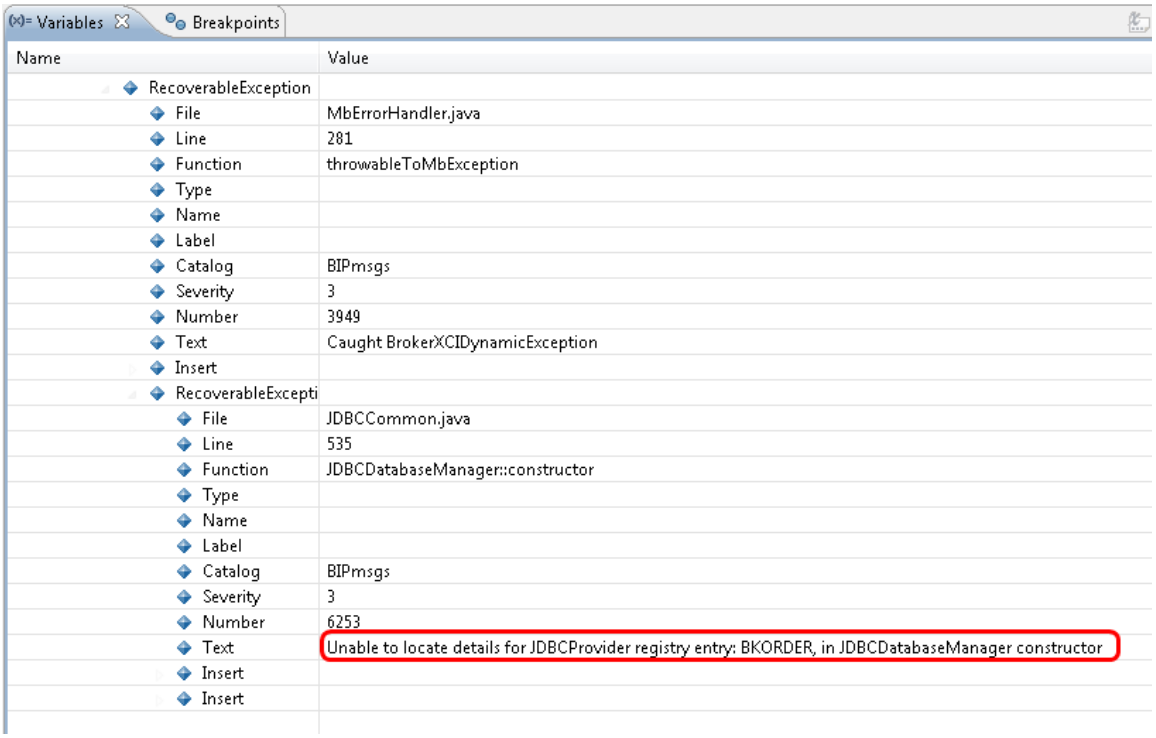
To find out the cause of the Exception, make sure breakpoint is set after the node on the Failure flow. Resume the suspended thread and investigate the ExceptionList at the breakpoint

Figure 4.46 ExceptionList in Variables



The exceptions are wrapped in a hierarchy and you will want to navigate to the last RecoverableException. As seen in Figure 4.47, the root cause is "Unable to locate details for JDBCProvider registry entry: BKORDER, in JDBCDatabaseManager constructor". With this information, it is clear that to resolve the issue is to create a JDBCProvider configurable service named BKORDER in the integration node.

Figure 4.47 Root cause of Exception



As seen in this section, the visual flow debugger from the Integration Toolkit provides an easy debugging experience. From the tool kit, the debugger can attach to the integration server JVM on a specified port. Breakpoints can be added without making changes to the message flow to examine the message or code variables. For an environment where the tool kit is connected to a remote server, there is no need for access to the server machine to debug the flow.

Pattern 5 – Business Process Manager Book Order process

This pattern shows how analytics information can be combined with operational decision management techniques in an IBM Business Process Management (BPM) process flow.

The example provided illustrates this pattern by using analytic data to supplement user-provided input in a business process to price book order transactions.

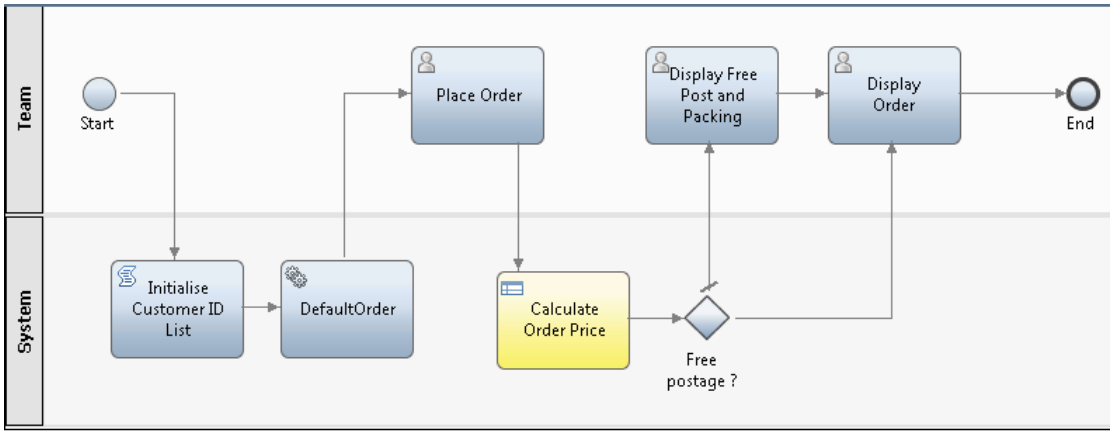
The pattern uses a BPM toolkit to create a decision service with a series of integration nodes to perform the various steps required. This toolkit can then be used in a process application, allowing the overall BPM decision service with those nodes to be safely shared by other business processes.

Process Overview

IBM Business Process Management provides a variety of methods to augment user-provided input to make a better-informed pricing decision. In this example scenario, a decision service (**Calculate Order Price**) retrieves customer information from a database and calls a scoring service to rate the probability of a customer accepting an offer before passing this data to IBM Operational Decision Manager to make an informed decision.

This decision service, and associated data types, is implemented within a BPM toolkit. This toolkit is then added to a process application, and the Calculate Order Price decision service is placed within a process flow which first provides input to, then exposes the results of, the decision.

Figure 5.1 Book Order Application Process Flow



This process flow provides a simple example of invoking the decision service as might be undertaken in a call center where the process user enters the order details on behalf of a customer.

At the start of the process, the list of customer IDs is initialized. In this example, this list is hard-coded, but could easily be replaced with a database or LDAP lookup or provided as an input to the process.

Next, a **Default Order** is initialized. This sets up a *BookOrderType* variable, **order**, with some sample data, which is further populated with the information needed to make a decision later in the process.

The **Place Order** node is a Human Service. This screen is where the order is created, the quantities set and any promotional code entered. The customer is selected by ID from the list populated in the first node. This ID is stored in the **customer ID** field of the order.

The **Calculate Order Price** node contains the ODM decision service, and the integration nodes needed to obtain prerequisite information. This is described in more detail in the next section.

Finally, the priced order and discounts are displayed to the user in **Display Order**, preceded by a message notifying of free postage and packaging if applicable. This illustrates how the output of the decision service can be used to influence the process flow.

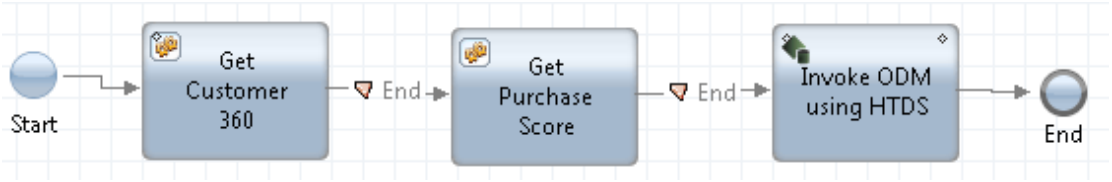
Calculate Order Price Decision Service

The Calculate Order Price (COP) node is a BPM decision service that coordinates the integration services and ODM decision service call necessary to make the decision. It is implemented within a BPM tool kit, which can then be used by process applications to integrate the decision service into their processes.

The COP service is split into three main stages, as can be seen in Figure 5.2:

1. **Get Customer 360** uses the customer ID to retrieve 360° information
2. **Get Purchase Score** passes the 360° information to the SPSS scoring service
3. **Invoke ODM using HTDS** uses the customer score and order details to calculate offers and order total.

Figure 5.2 Decision Service Integration Flow



First, the COP service takes the **customer ID** and performs a database lookup to retrieve additional 360° information about the customer. This 360° information is used in the scoring service and also passed back to the process.

Next, the node calls an Integration Service which invokes a Java method. This method extracts the loyalty and gender from the customer 360° information and performs a call to the SPSS scoring service to retrieve the likelihood that the customer will eventually place an order.

Once the 360° data and score have been retrieved, the ODM decision service itself is invoked to calculate the price of the order and any discounts which apply. This decision service is deployed to an IBM ODM Decision Server.

To integrate BPM with an ODM decision service, the data model from the decision service must be created within BPM. This can be done manually, but it is better to import the data types from the service API as described in the next section. These data types can then be used within the process and within this decision service.

The decision service integration will expose three of the imported types as inputs and/or outputs as shown in Figure 5.3:

- *BookOrderType*

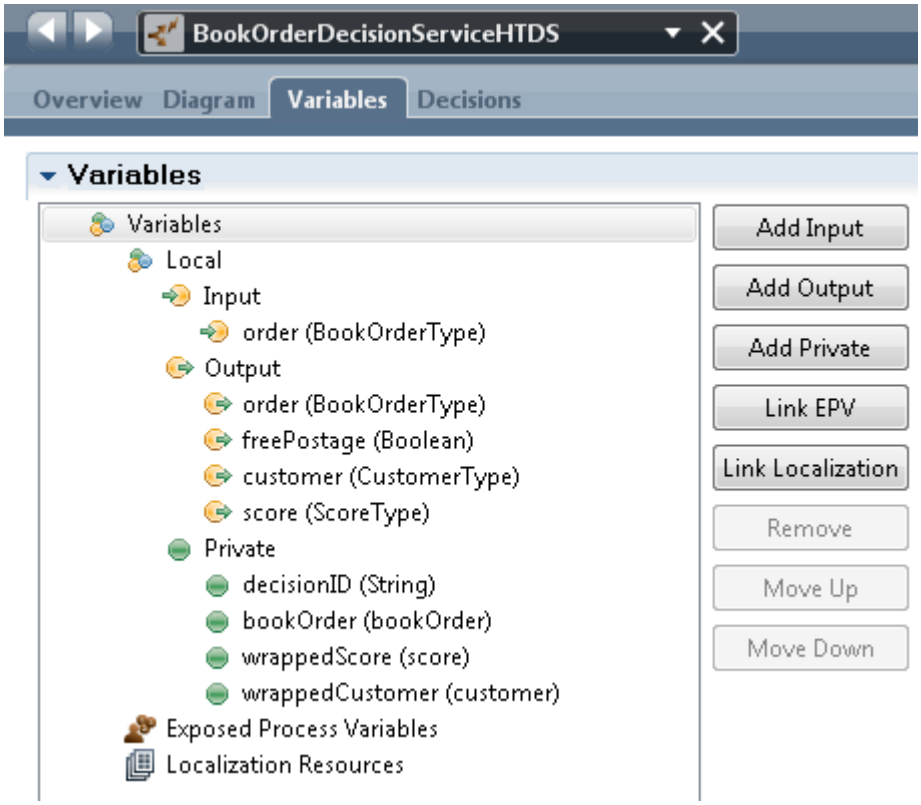
- *CustomerType*
- *ScoreType*

The COP service takes a single input, **order**, which is a *BookOrderType*. This variable contains the order details, including which books are being ordered and the ID of the customer placing the order. This variable is also an output variable; the COP service will augment the order with offer and postage information.

The **score** and **customer** outputs will be populated by the 360° information lookup and scoring service respectively, and are available to supplement the human service which displays the order at the end of the process. A final output, **freePostage**, is a *Boolean* which represents whether the order includes free postage and packing or not. This is set as part of the decision service post-processing.

The three private variables **bookOrder**, **wrappedScore** and **wrappedCustomer** are used only for invoking the decision service. These should have default values set allowing the real input and output variables to be easily mapped. The **decisionID** is used to identify individual decisions in the decision warehouse. This might be set to an order number in a real solution..

Figure 5.3 Decision Variables



The customer ID is extracted from the order and passed to **Get Customer 360** which returns a populated *CustomerType*. This is stored in **customer** and then passed to **Get Purchase Score**, which calls SPSS and outputs the **score** variable.

These three variables are then passed to the ODM decision service which updates the order variable returned to the process as described in the next section.

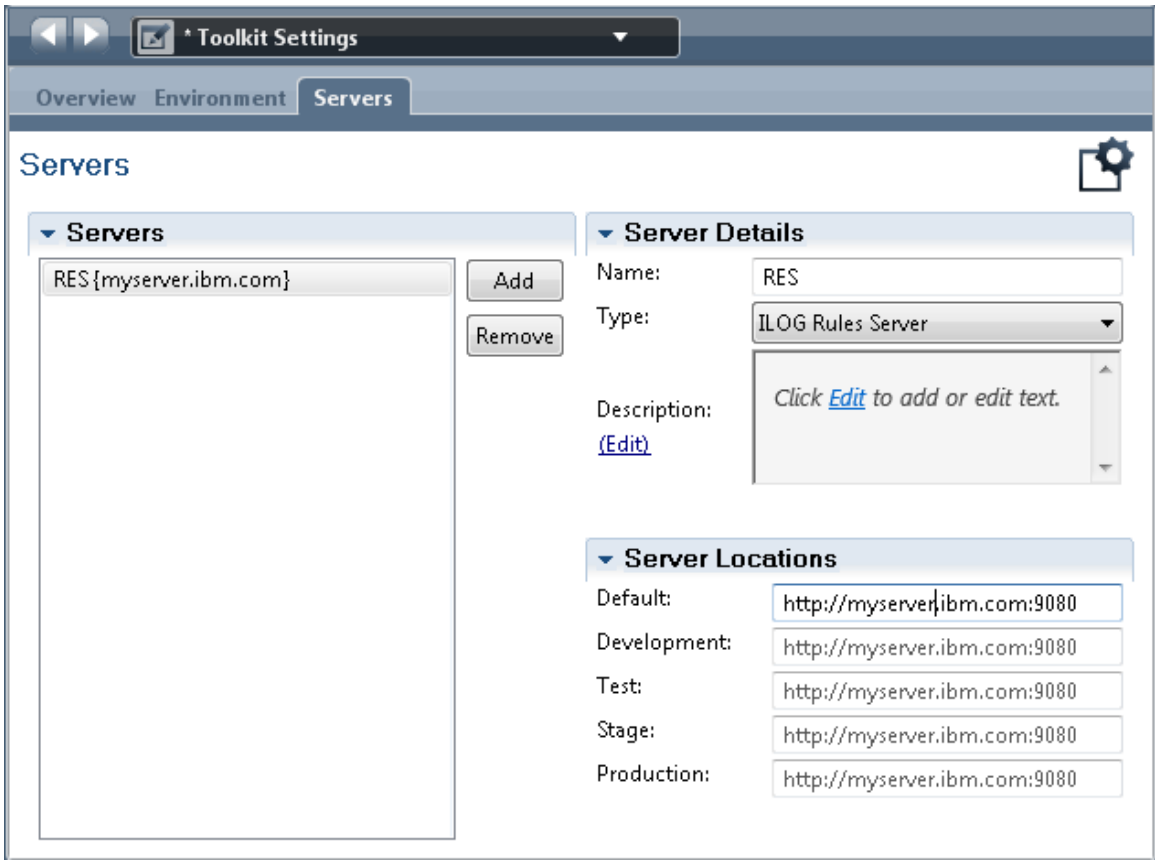
ODM Decision Service Integration

To integrate the ODM decision service, the data types must be created and the server configuration defined.

The server is defined on the **Servers** page of the **Toolkit Settings**. To configure a server:

- Select the settings of the toolkit
- Select the **Servers** tab
- Click **Add**
- Enter a **Name** for the server. This is used within your application, but does not need to correspond to a server name
- Set the **Type** to **ILOG Rules Server**
- Enter a **Description**, if desired
- Enter the **Server** locations as appropriate. If using only a single ODM server, this can be entered in the **Default** field and the remaining fields will be auto-filled. In a multi server topology, you should set the decision server locations for each environment.

Figure 5.4 – RES configuration in BPM toolkit



With the server defined, the decision service node **Invoke ODM using HTDS** can be configured. This is a JRules Decision Service node, and it is configured using the **Implementation** properties page. The server defined above is selected, and set the **SOAP Port**, **Username** and **Password** fields are set as appropriate. Clicking **Connect** populates the **Rule App** and **Ruleset** drop-downs from the rule execution server.

In this scenario the BookOrderDecisionService (RuleApp) and BookOrderPricing operation (ruleset) described in pattern 1 have already been deployed to the Decision Server and can be selected from the drop downs.

When selecting the versions you should select a specific version for the RuleApp as this will normally be used to determine the Business Object Model on which the ruleset is based. Updates to the ruleset that change the underlying eXecution Object Model or schema will not be compatible. It is good practice to select the latest version of the ruleset which means that updated rulesets will immediately be used when the decision is next called from the process.

Clicking **Generate Types** brings up a wizard, which interrogates the rule execution server to retrieve the WSDL for the decision service and generates all types required within the toolkit, such as *BookOrderType*, *CustomerType* and *Score*.

Figure 5.5 – Configuring the ODM Decision Service

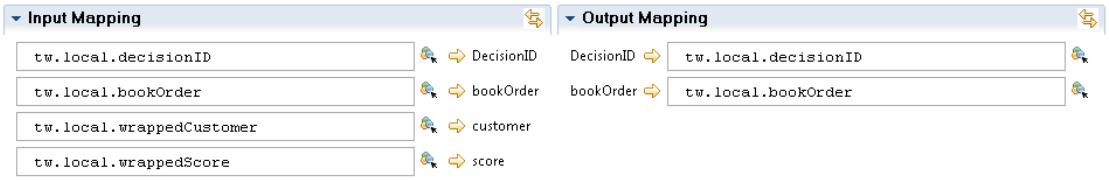
The final stage before the ODM decision service can be invoked requires the order, score and customer to be wrapped into parent objects to match the decision service parameters. This is done using the pre-execution assignments, where the **freePostage** variable is also initialized to *false*. Failure to initialize **freePostage**, or to wrap the other variables will result in a *NullPointerException*. The **DecisionID** can be optionally left as *null* or a blank String. In this event, the decision ID is assigned by ODM.

Listing 5.1 Pre-execution assignments for decision service

```
tw.local.bookOrder.bookOrder= tw.local.order;
tw.local.wrappedScore.score = tw.local.score;
tw.local.wrappedCustomer.customer = tw.local.customer;
tw.local.freePostage = false;
```

The decision service takes the order, score and customer along with the decision ID. The result is mapped back to the **bookOrder** variable. The decision ID is also stored back into the **decisionID** variable. If this was left blank, the value assigned to the execution by ODM will be returned here.

Figure 5.6 Input and output mapping for decision service



After the decision service has executed, the *BookOrderType* is extracted from **bookOrder**, and the **freePostage** flag calculated based upon the post and packaging field of the order summary.

Listing 5.2 Post-execution assignments for decision service

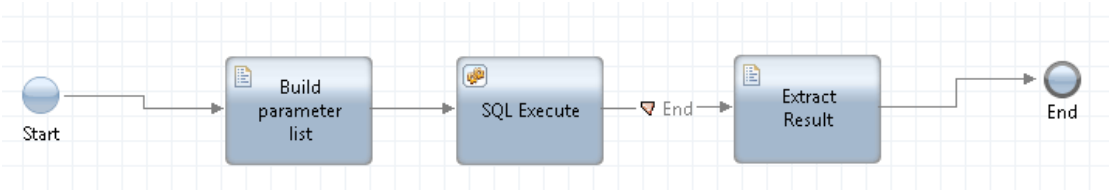
```
tw.local.order=tw.local.bookOrder.bookOrder;
if (tw.local.bookOrder.bookOrder.orderSummary.postAndPackaging == 0)
    tw.local.freePostage = true;
```


360° Data Retrieval

Before the SPSS scoring service can be invoked, information about the customer needs to be retrieved. The service scores based upon the gender and loyalty of the customer, and this 360° information is stored in a DB2 database.

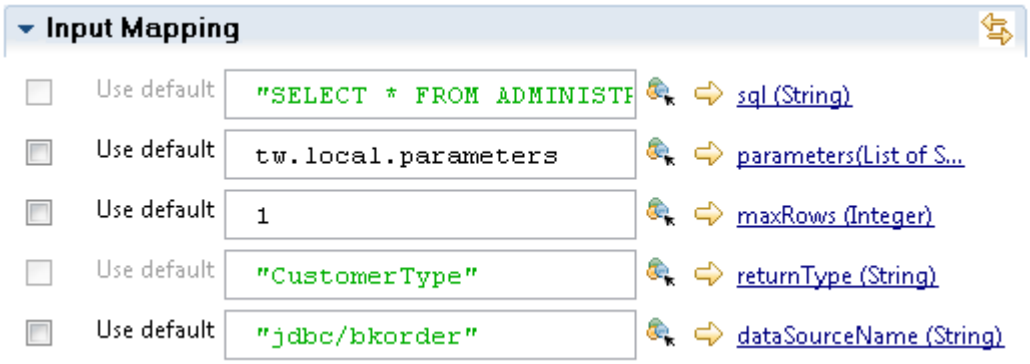
The 360° information retrieval is implemented as an integration service. The service has a single *String* input variable, **customerID**, and a single output variable, **customer**, which is a *CustomerType* variable. The service flow can be seen below.

Figure 5.7 360 Data Lookup Integration Service



The **SQL Execute** node uses the BPM standard service **SQL Execute Statement**. This takes as input both configuration for the data source and the SQL to execute.

Figure 5.8 360 Data Lookup Integration Service



The **dataSourceName** must match a data source configured on the WebSphere Application Server that BPM is installed on. In this example, **jdbc/bkorder** is a DB2 data source with appropriate credentials.

sql, **parameters** and **maxRows** all configure the SQL to be executed. To include parameters to the SQL statement, use the ? character. These will be substituted using entries from the **parameters** list. To retrieve the customer information, the following query is used:

Listing 5.3 Customer 360° Retrieval SQL

```
SELECT * FROM ADMINISTRATOR.CUSTOMER WHERE CUSTOMERID=?
```


This allows the customer ID to be included as a parameter, rather than manually combined into the SQL string. The parameter list is initialized in the **Build parameter list** node. This is a Server Script node, with an implementation as follows:

Listing 5.4 Customer 360° Retrieval SQL

```
tw.local.parameters[0] = new tw.object.toolkit.TWSYS.SQLParameter();
tw.local.parameters[0].mode = "IN";
tw.local.parameters[0].type = "VARCHAR";
tw.local.parameters[0].value = tw.local.customerID;
```

The list is initialized with only one parameter: the customer ID. This is taken from the **customerID** input.

The final input, **returnType**, configures what type of BPM data type the results will be created as. If this is set to a user-defined data type, columns from the returned results will be mapped to the attributes of the data type with the same name (case insensitive). By setting this to **CustomerType**, the output of the **SQL Execute** node will be a list of *CustomerType* objects.

As there will only be a single customer per customer ID, the **Extract Result** Server Script sets the **customer** output variable to the first element in the results list:

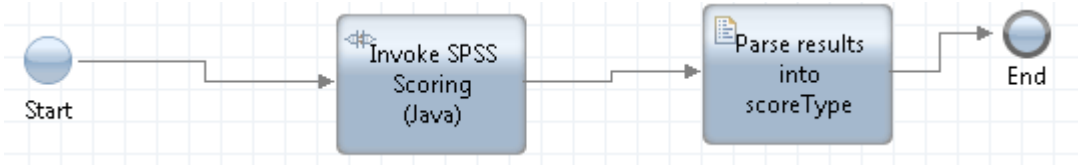
Listing 5.5 Customer 360° Retrieval SQL

```
if(tw.local.results.listLength == 1){
    tw.local.customer = tw.local.results[0];
}
```

Scoring Service

The **Get Purchase Score** node is another Integration Service. This service uses a Java Integration node to invoke a small Java method **getScore**, which in turn uses the JAX-WS client (described in pattern 3) to call the SPSS scoring service.

Figure 5.9 Scoring Service Flow



The service takes a single input – a *CustomerType* object – and returns a single output: a *ScoreType* object. The **loyalty** and **gender** are extracted from the **customer** and passed to the scoring service. The result from the scoring service is parsed into the **score** and returned from the integration service.

Listing 5.6 getScore Java Method

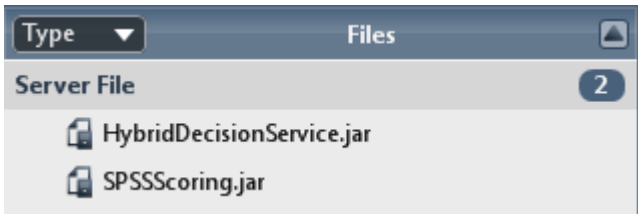
```
public ScoreType getScore (String loyalty, String gender){
       return getScore(new ScoreType(loyalty, gender, null, 0));
}
```



```
public ScoreType getScore(ScoreType score) {

        //Create a new score request
        ScoreRequest scoreRequest = new ScoreRequest();

        //Define the PredictedLoyalty configuration
        scoreRequest.setId("OrderPlacedPrediction");

        //Create the table and first row
        RequestInputTable rInputTable = new RequestInputTable();
        rInputTable.setName("OrderPlacedScoring");
        RequestInputRow rInputRow = new RequestInputRow();

        //Add the row and table to the request
        rInputTable.getRequestInputRow().add(rInputRow);
        scoreRequest.getRequestInputTable().add(rInputTable);

        //Setup the parameters
        Input loyaltyi = new Input();
        loyaltyi.setName("loyalty");
        loyaltyi.setValue(score.getLoyalty());
        rInputRow.getInput().add(loyaltyi);

        Input genderi = new Input();
        genderi.setName("gender");
        genderi.setValue(score.getGender());
        rInputRow.getInput().add(genderi);

        //Invoke the scoring service
        ScoreResult scoreResult = null;
        try {
               scoreResult = service.getHttpV2().getScore(scoreRequest);
        } catch (Exception e) {
               e.printStackTrace();
        }
        //Get the first Row of values
        RowValues values = scoreResult.getRowValues().get(0);
        //Get the predicted loyalty column 3
        score.setOrderPlaced(values.getValue().get(2).getValue());
        //get the confidence column 4
        score.setConfidence(Double.valueOf(values.getValue().get(3).getValue()));

        //return the results
        return score;
}
```

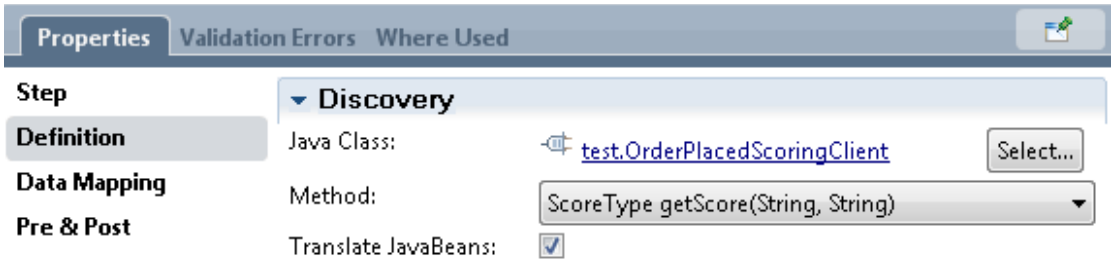
To use the SPSS invocation code, it must be uploaded as a server file to the BPM toolkit. The invocation code depends on the JAX-WS client, which needs to be on the class path, so this must also be uploaded as a server file.

Figure 5.10 Toolkit Server Files



Once the JAR has been uploaded, the **getScore** method is available for use in a Java Integration node.

Figure 5.11 Selecting the getScore Method



By selecting **Translate JavaBeans**, the return type of the method is parsed into an *XMLElement* which is easily processed to extract the result. Data mapping for this method is straightforward, with the two arguments (seen in Listing 5.6) being **loyalty** and **gender**.

Pattern Testing

As this pattern uses a process application with UI screens to expose the decision service, it can be tested simply through the web UI of BPM. In this section, the process is executed with two separate customers as inputs and the differing results of the decision service are displayed. Advice on debugging is also described.

The process can be started from within Process Designer by clicking the play icon in the top-right corner of the editing window:

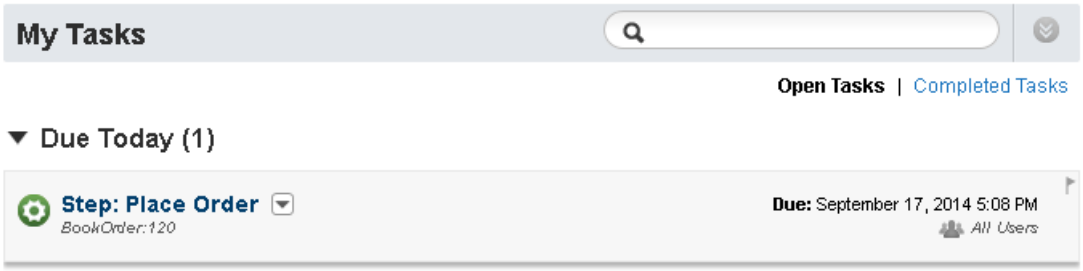
Figure 5.12 Starting the process

This switches Process Designer into inspector mode. From the inspector mode you can either select an active task to execute/debug or select an active process and launch the web UI from the upper-right icon:

Figure 5.13 Starting the UI

If not already logged in, enter credentials at the log in prompt, and then select the **Place Order** step from the list of tasks.

Figure 5.14 Selecting the task

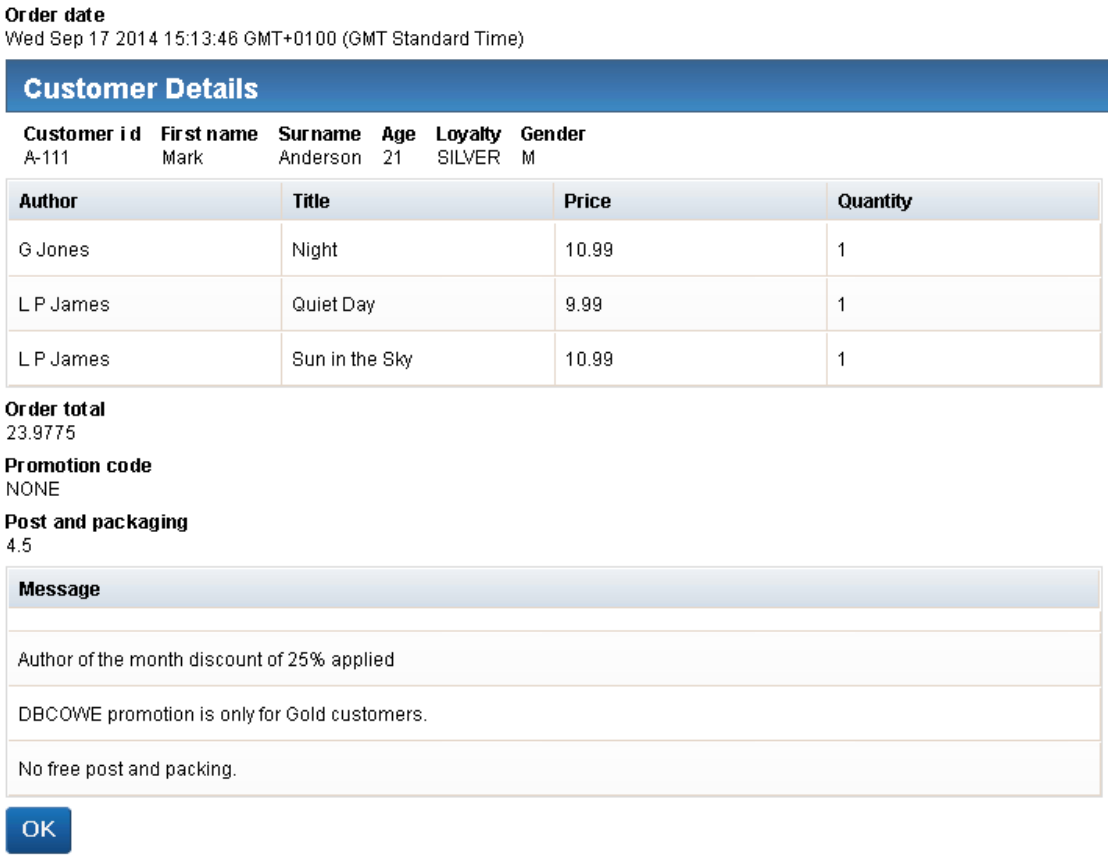


This claims the task for the user, and brings up the first UI screen to create the order.

Figure 5.15 Creating a sample order

Despite the entry of the **DBCOWE** promotion code, the discount is rejected as customer **A-111** is not a gold customer. The customer 360 information that is returned from the decision service is also displayed back as part of the summary:

Figure 5.16 Display order screen

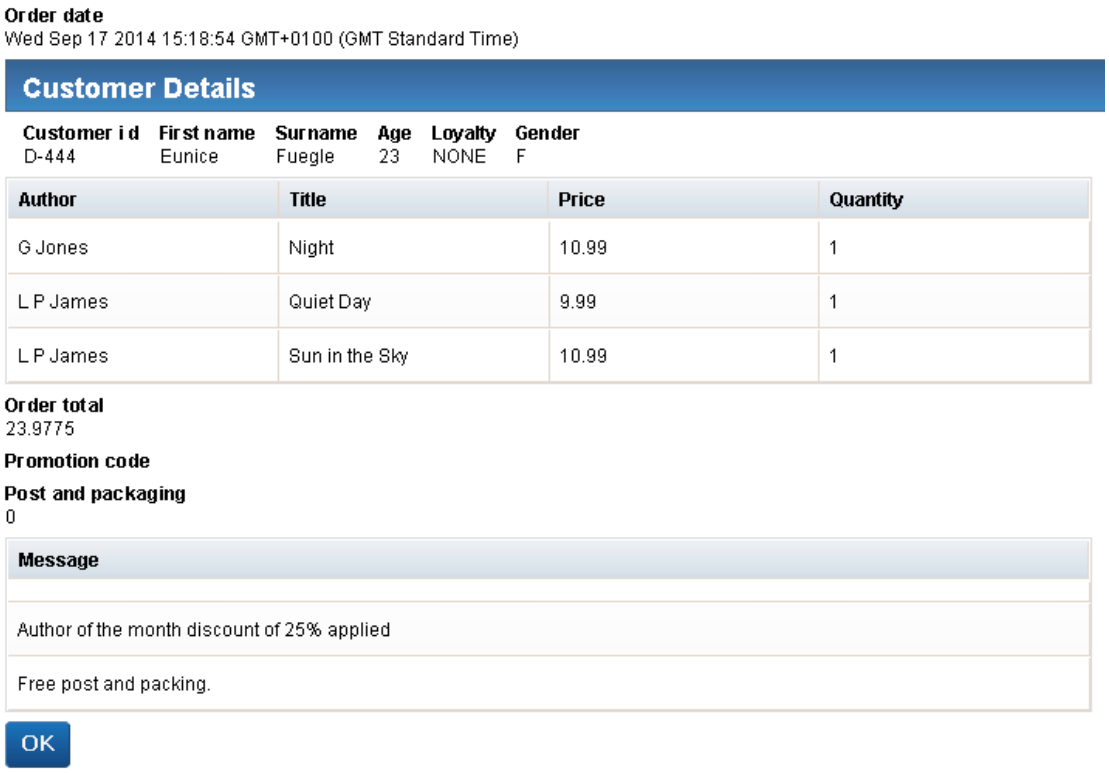


If you select Customer **D-444**, **Eunice Fuegle**, and no promotion code, she is unlikely to purchase and is offered free P&P. The quote is routed to the extra human service to display this notice.

Figure 5.17 Free post and packing screen

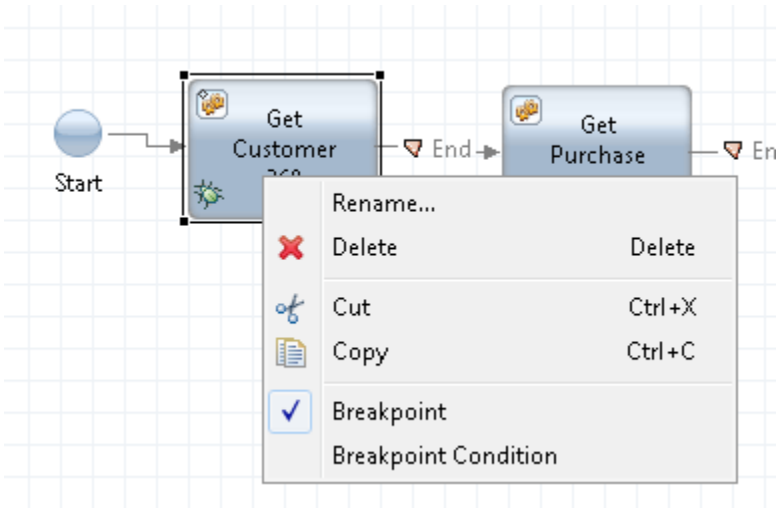
After the notice is displayed, the order summary is presented as before:

Figure 5.18 Display order screen with free P&P



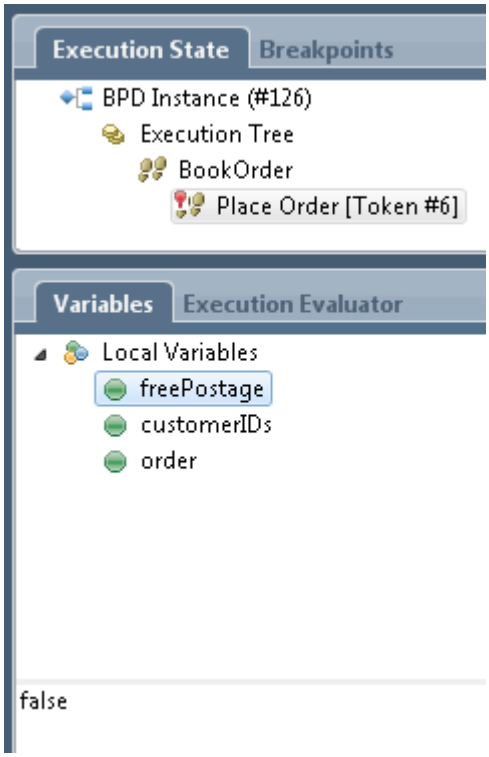
In the event that the decision service is not executing as expected, BPM debugging features can be used to step through a process or service to identify the state of variables at various stages. Before executing a service from within Process Designer as normal, identify the stage which is producing unexpected results. Open the service in the editor, and select the relevant node. Right-click on the node, and select **Breakpoint**, as show below:

Figure 5.19 Enabling breakpoints



Once breakpoints are set, execute the service as before. When the node on which you have placed a break point is reached, the execution will pause, allowing you to examine the state of any variables at that point:

Figure 5.20 Inspecting variable values



Alternatively, you can step through each stage of a service in turn by selecting the **Debug** icon next to the play icon. This loads the web-based debugging screens, displaying the value of every variable and providing buttons to progress through the service.

Figure 5.21 Starting a service in debug mode

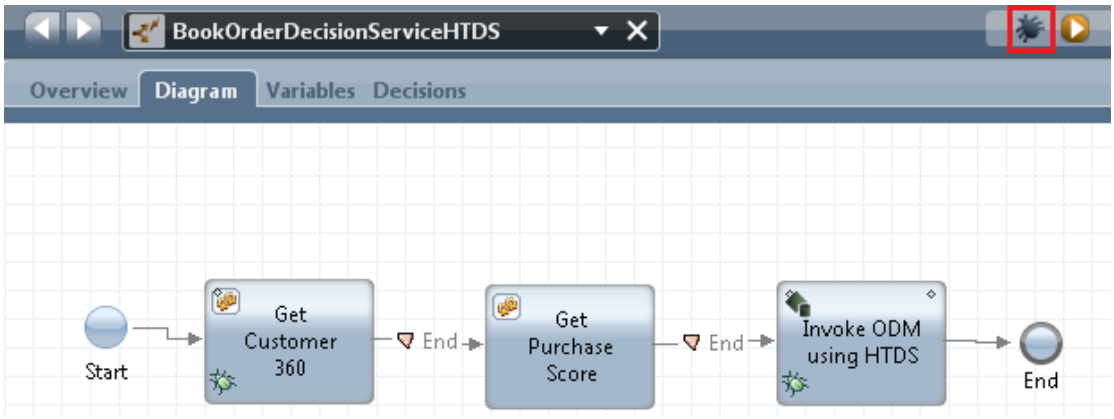
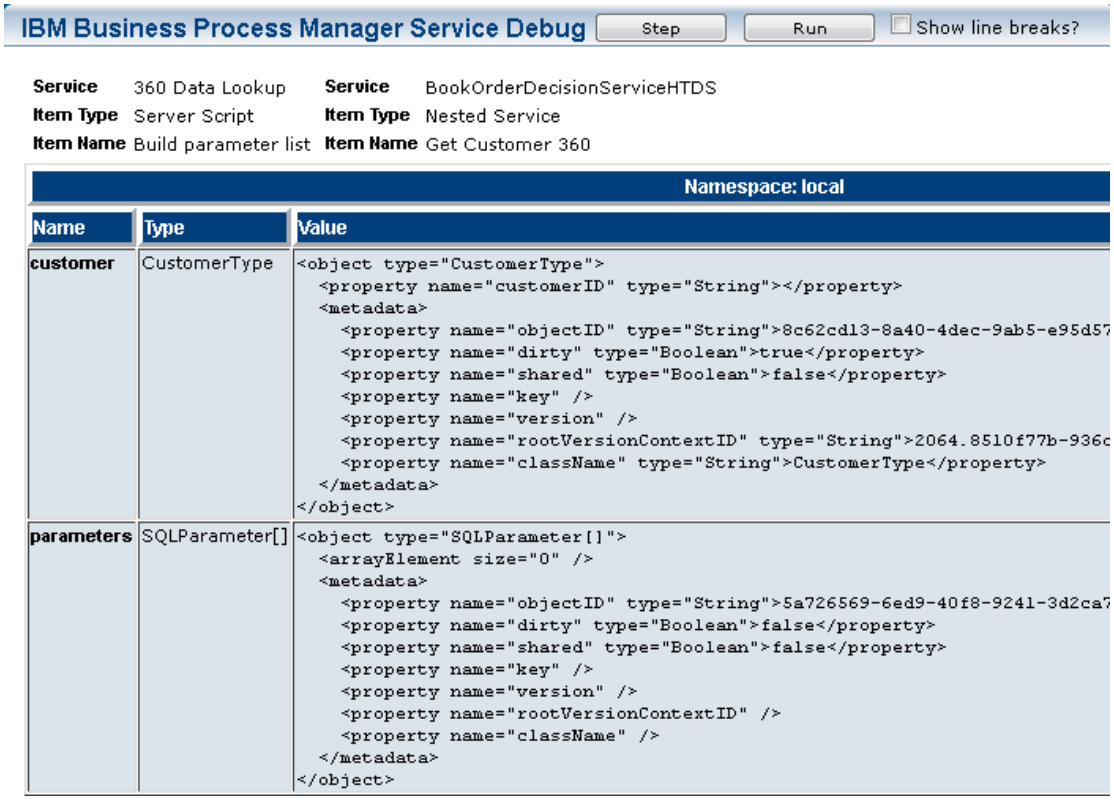


Figure 5.22 Using the web-based debug interface



If it appears that the correct data is being sent to the ODM decision service, it may be helpful to examine the execution trace in the decision warehouse to ascertain if it is being correctly received and processed. This is accessed from the web console of the rule execution server. It displays the input and output parameters of each decision execution, along with other statistics such as the number of rules that were executed.

The rule execution server console and decision warehouse are described in further detail in the Process Testing section of pattern 4.

If you require more information on BPM debugging, this is documented in depth in the IBM Knowledge Center.

Conclusion

This article has examined how Analytical and Operational Decision Management techniques can be used together in a solution. The goal of this integration is to leverage the emerging business insight obtained from analytics in the operational decisions made on a regular basis in the solution. This allows those decisions to respond dynamically to the evolving market providing an optimized response in the complex emerging business environment.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM
product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England SO21 2JN

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply
to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this
publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England SO21 2JN

Such information may be available, subject to appropriate terms and conditions,
including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on
generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business
enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be
liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp.
Sample Programs. © Copyright IBM Corp. _enter the year or years_.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at `www.ibm.com/legal/copytrade.shtml`.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.