IBM Operational Decision Manager

Version 8.6.0

Patterns for Operational Decision Management in Streams

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.

Title:
Patterns for Operational Decision Management in Streams.

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 Operational Decision Management decisions to be integrated into Big Data streams processing to improve the business insights and situational awareness used to produce actionable responses in business solutions.

A retail scenario illustrates how IBM Operational Decision Manager can be integrated into InfoSphere streams to help analyse customer book buying behaviour and thus improve the chance of a successful order.

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.

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

Technology:
Operational Decision Manager 8.6
Infosphere Streams 3.2.1

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.

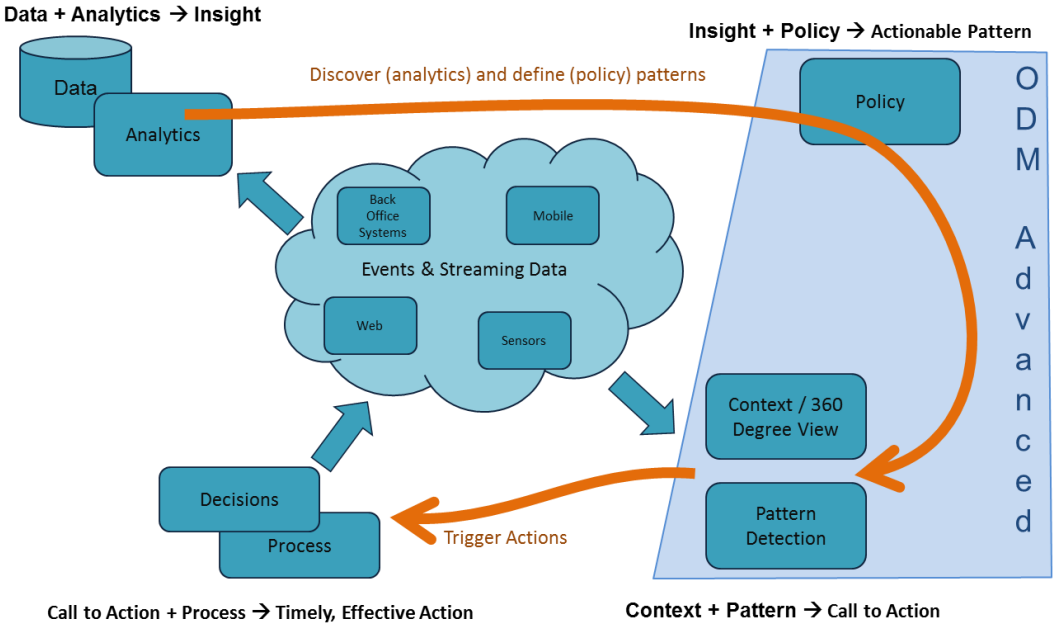
Traditional approaches focused on recording and manipulating records using synchronous decision services to define the actionable response as part of a well-defined process. With the advent of mobile, cloud and big data processing, businesses need to respond to emerging risks and opportunities at the 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, 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 are 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 – shows the basic patterns for integrating operational decisions into solutions
- Patterns for operational and analytical decision management in Smarter Processes - shows how operational decisions and predictive analytics can be leveraged in solution
- Patterns for integrating operational decisions into streams and Big Data solutions (this article) - shows how operational decisions can be leveraged as part of an analytical insight solution
- Patterns for actionable insight 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

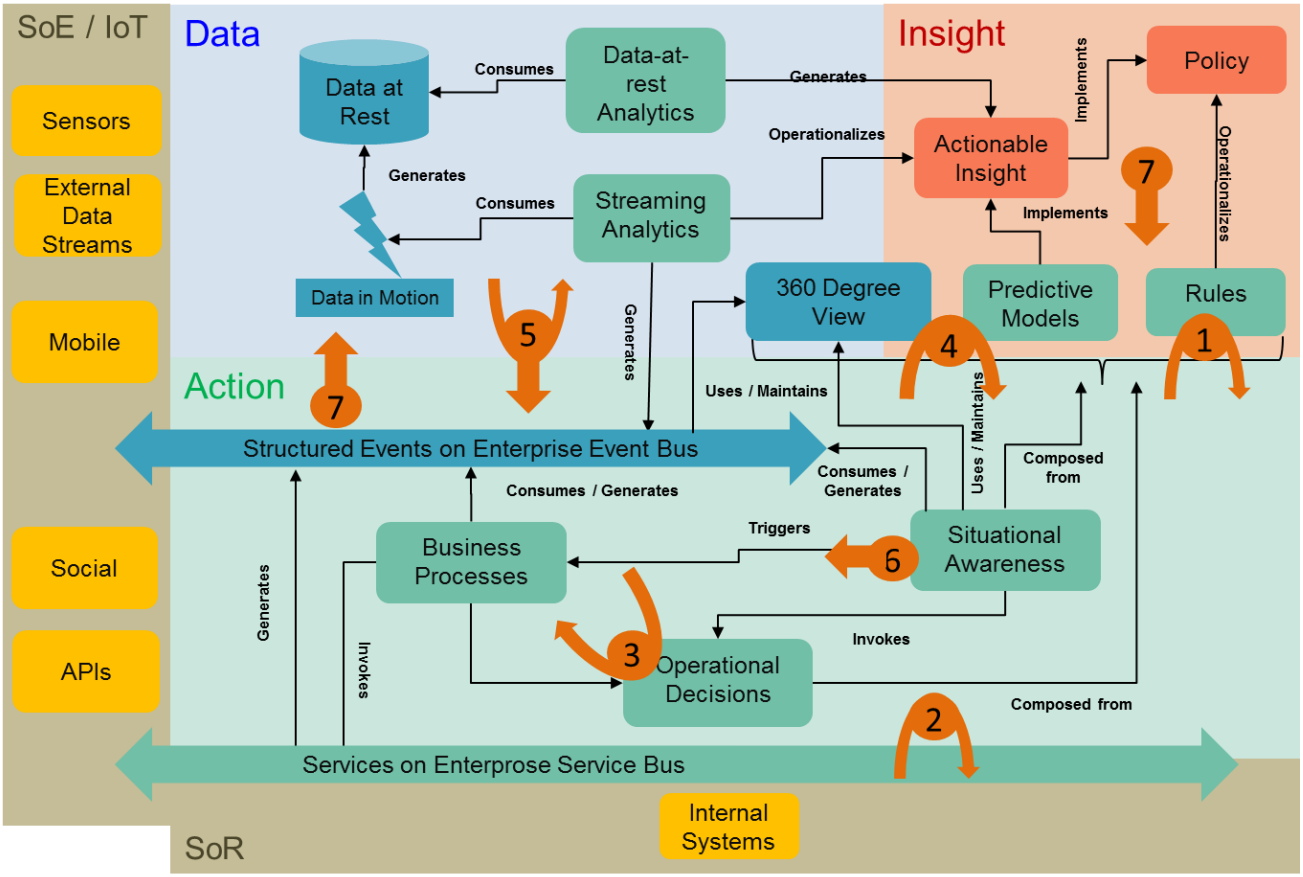


Figure 2 shows five main areas of actionable insight solutions.

- 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 learnt 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 other articles:
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 decision services allowing a more accurate decision at the time the decision is made.

This article concentrates on:
5. Applying rules based decisions in Big Data streams based processing

The following integration points will be described in other articles:
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 (e.g. 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.

Readers should refer to the IBM SPSS Collaboration and Deployment Services Information Center for further information on the IBM SPSS Analytics product.

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 (processing data-in-motion) and Hadoop (processing data-at-rest) 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 using rules is becoming more and more important to identify emerging situations that require attention. This article concentrates on how ODM is integrated with InfoSphere streams.

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

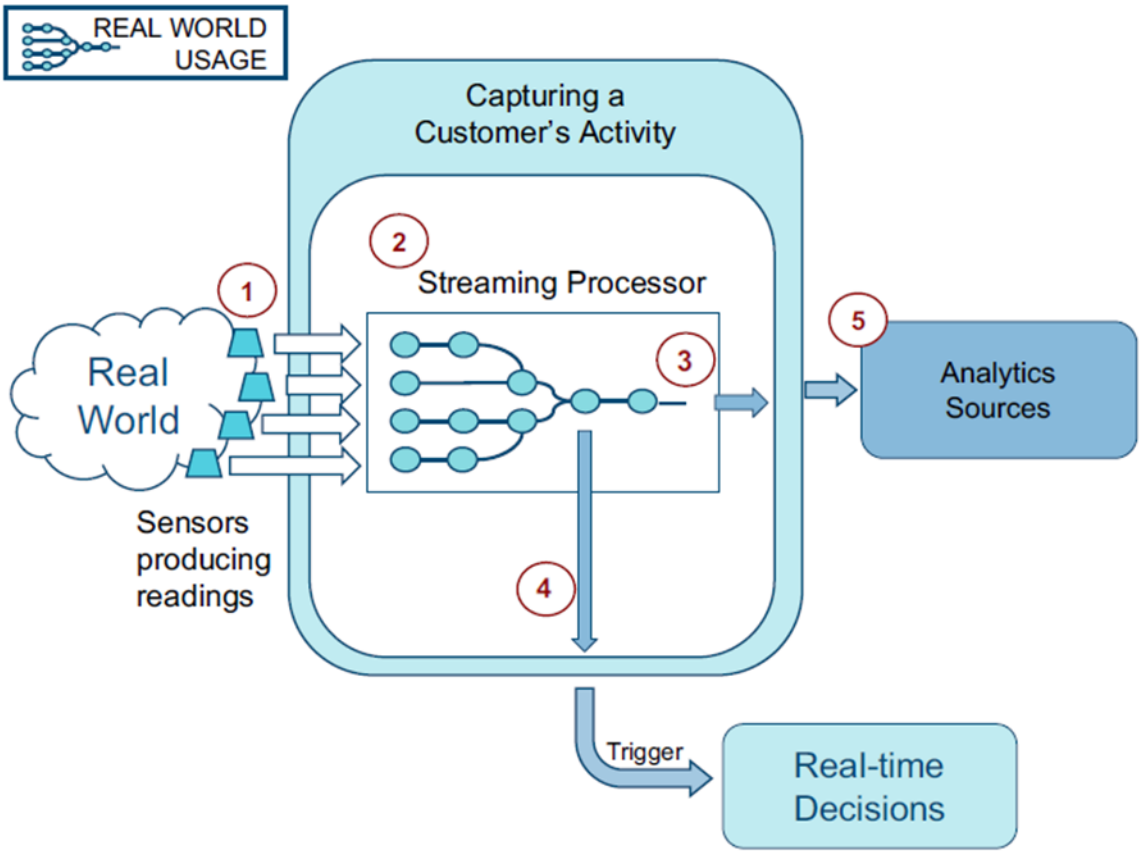
The key goal of decision services and complex rule based event processing is to allow the business to express and manage the required behavior of their solutions using rules. By monitoring and analyzing the behavior of the business in response to the decisions, organizations can understand how the rules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

Infosphere Streams Integration Overview

Streams applications process large volumes of moving data in data streams and are created by using the Streams Processing Language (SPL). In this integration pattern we will start with a typical streams solution as shown in Figure 3.

Figure 3 Overview of typical streams integration



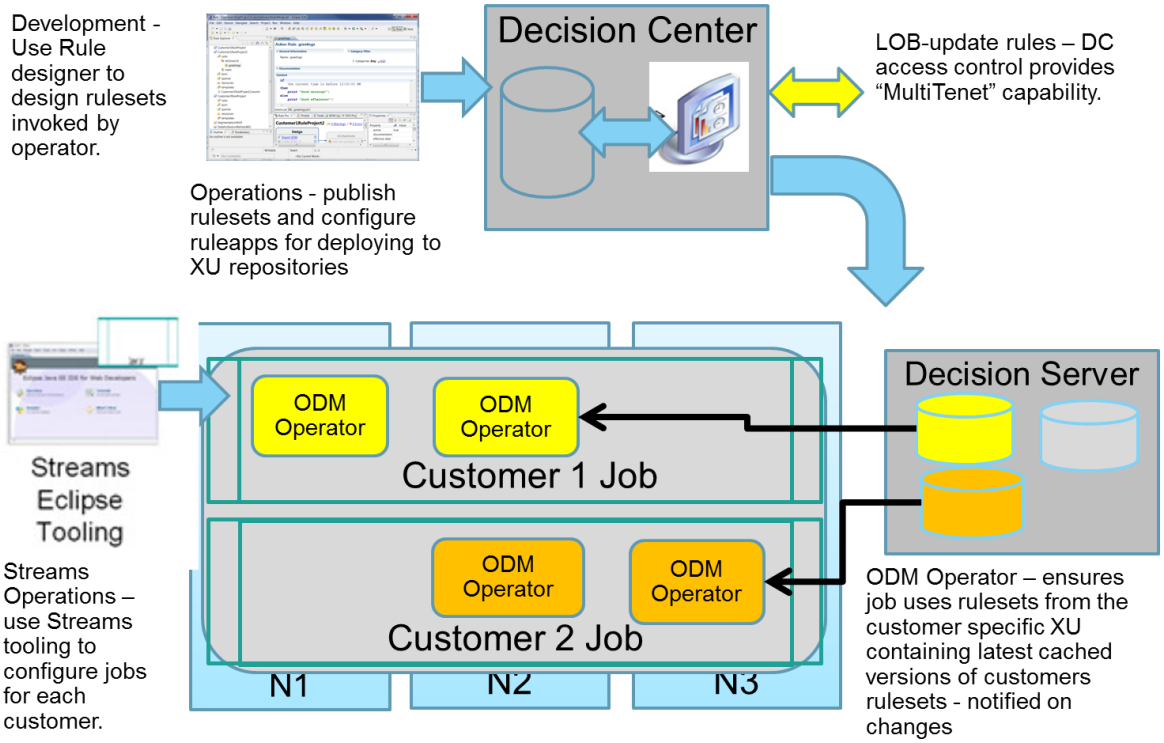
In the diagram above there are five key interactions taking place within an Infosphere Streams environment:

1. You can use the Streams Processing Language (SPL) to define data sources for most devices, sensors, or application systems. To simplify application development, you can use the predefined adapters or toolkits that are included with the product. This results in streams of similar information arriving from disparate sources. A data stream is a running sequence of tuples which can encompass both structured and unstructured data.
2. Streams processing provides the technology for "real-time" analytics for this "data-on-motion". This allows a wide range of characterization and filtering of the data as well as advanced analytics through the use of SPSS, R, Time-series analysis or other toolkits or operators. This article describes how the rules toolkit (and ODM) can be used in these streams applications.

3. The Streaming Processor is arranged as a distributed pipeline of processing nodes and operators. The main components of SPL applications are tuples, data streams, operators, processing elements (PEs), and jobs. By combining operators and streams together into a streams application and deploying that application across multiple machines (processing elements), a very high performance processing pipeline can be developed.
4. Whenever an operator receives a tuple from an input stream, the operator produces modified tuples on output streams. Toolkits are packages of SPL artifacts that are designed for specific business environments or activities. The Rules toolkit allows ODM rulesets to be executed as an operator in the stream. This allows the stream tuples to be classified or filtered by the rules to identify tuples (or events) of significance in other decision making systems. This would include ODM Decision Server Insights that would allow these more significant events to be interpreted in a broader longer-term context.
5. The resulting streams data can be consumed in many ways with the toolkits or fed into other products such as Hadoop, BI or other data warehouse technologies for doing deeper data-at-rest analytics, and looking more at historical results. This data at rest insight can be leveraged as scoring services or encoded as rules and brought back into the real-time decision making process making the insight immediately actionable.

The Ruleset Executor Node can be configured in the streams as shown in figure 3. Each Ruleset Executor Node works with an embedded JSE Rule Execution Server deployed onto the Java Virtual Machine in the Streams Node. It is often the case that a streams topology will need to support different streams using different rulesets with potentially different owners. Figure 4 below shows how the ruleset management fits in with the streams architecture

Figure 4 Typical integration of Streams with IBM Operational Decision Manager



The streams tooling allows jobs to be defined that process the information through a number of operators including the Ruleset Executor operator. The ruleset executor operator requires a JSE Rule Execution Server to be deployed on the JVM of each node (machine) that the operator will run on (N1-N3). The operators can be configured to download their rulesets from separate Decision Server repositories. This allows the rulesets for different customers to be isolated if required.

Providing a Decision Server to manage rulesets also means that updated rulesets can be deployed from Decision Center according to governance best practices. It also means that the JSE RES's embedded in the streams can be monitored and managed from the Decision Server console. For a streams application that is running with very low latency, it would not normally be appropriate to use a decision warehouse so the decision server would mainly be used for testing, simulation and managing deployed rulesets.

Article Overview

The remainder of this article describes the following aspects of ODM and streams integration

Business Scenario and Information models

This describes the example retail scenario that illustrates how IBM Operational Decision Manager can be integrated into Infosphere streams to help analyse customer book buying behaviour and thus improve the chance of a successful order. The section also describes the information models used in the scenario by both the streams application and OBM Operational Decision Manager.

Classification Pattern.

This section describes how an ODM JSE Rule integration server (RES) can be embedded into a streams operator allowing the stream processing to be configured dynamically by rules. In this case the stream contains the individual customer actions when building up a book order including viewing, adding or removing a book from the order. The rules classify each tuple in the stream which can then be used in later streams analytics processing.

Book Click Decision Service

This section describes how to develop an ODM decision service to be used in streams processing. The section starts by describing how to support the simple flat object model provided in the Classification pattern and goes on to describe how to structure and write rules that can leverage more sophisticated hierarchical object models that can be integrated into streams as described in the filtering pattern.

Filtering Pattern.

This pattern describes how an ODM JSE Rule execution server (RES) can be embedded into a streams operator allowing the stream processing to be configured dynamically by rules. In this case the Rule executor operator uses the rules to identify and filter certain tuples of interest. This pattern shows how to use custom mapping between ruleset parameters and the streams allowing the operator to provide events on a separate port that can then be routed to other systems for action.

Annex: Installation and Configuration

This section provides an overview of installing and configuring IBM InfoSphere streams with IBM Operational Decision Manager to support the use of business rules with streams applications. The tutorial is based on the use of the InfoSphere Streams Quick Start Edition which is available for non-production environments as a VMWare Image or as a native Linux install. The version used for this tutorial is v3.2.1. This section also includes instructions on running the ODM samples provided with IBM InfoSphere streams and configuring the JMS messaging toolkit to work with WebSphere MQ.

Business scenario and information model

This section describes a simplified retail scenario that illustrates how IBM Operational Decision Manager can be integrated with a range of other IBM products across the

Smarter Process, Connectivity and Analytics portfolios. The goal of the scenario is to perform discounted pricing of books based on emerging customer characteristics. In this article we extend this scenario to examine how we can use IBM InfoSphere Streams with IBM Operational Decision Manager to help analyze individual customer web clicks

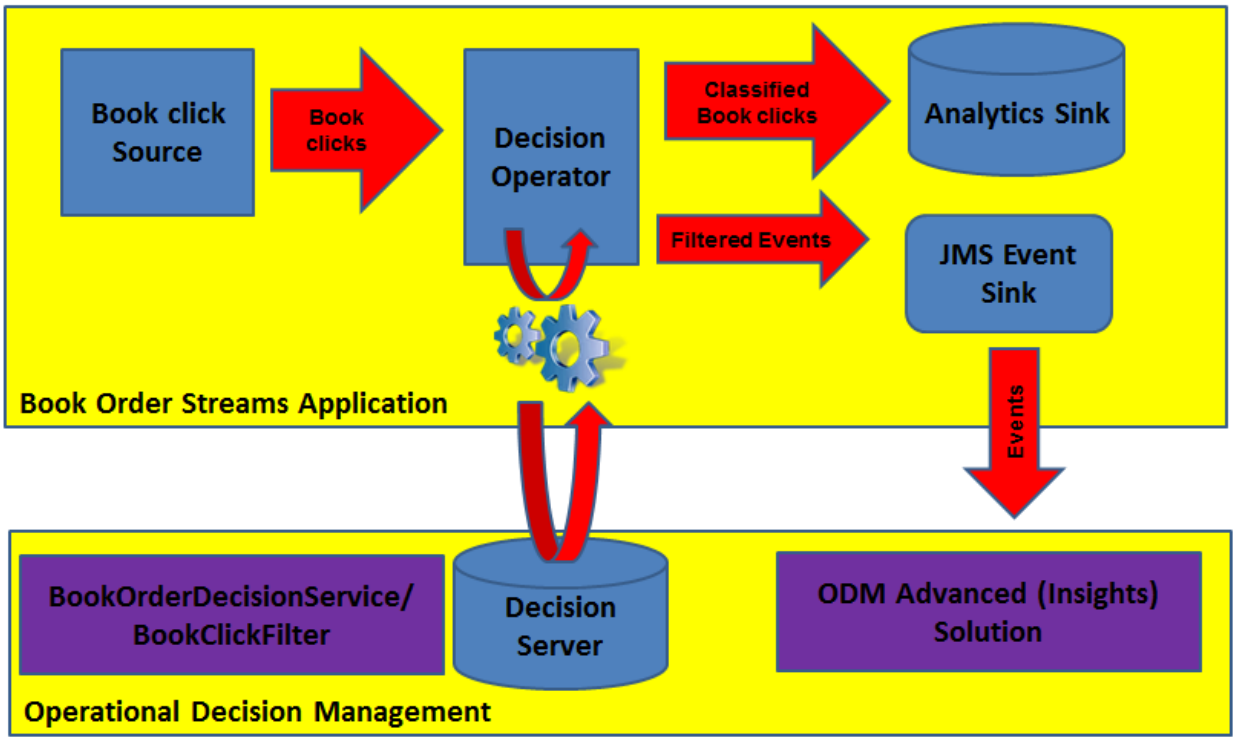
Scenario overview

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 main scenario described in previous articles 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.

In this article the scenario focusses on the click stream analysis that leads up to the customer requesting a quote.

A high-level overview of the solution is shown in figure 2.1 below:

Figure 2.1. Scenario Overview



In this scenario, customers browse through a number of web sites selecting books to add to a shopping basket. Each click that a customer makes (view a book, add a book to the basket, remove a book from basket etc.) is routed into a stream using a "book click" source node. Each book click contains key information about the customer and the book being viewed together with basic information about the state of the shopping basket.

This stream is then passed to an ODM operational decision that uses rules to classify each click so that it can be passed to analytics to better understand the buying patterns of customers. In addition, some clicks may imply a need for more detailed action – for example if a customer has a high value shopping basket – where the organization might want to start providing offers to close the deals. In this case the rules can be used to create a new event tuple that can be sent through a filtered events stream and routed into another system (such as ODM Advanced Insights) that allows the sequence of actions of that customer to be analyzed and appropriate actionable responses taken.

This article describes the integration between streams and ODM using the decision service capabilities provided in version 8.6. As an example we describe a BookClickDecisionService decision service that provides two operations (rulesets):

- **BookClickClassification** uses simple flat input and output parameters that map to each field in the tuple. The rules in this operation are those described in the Classification pattern and basically classify each tuple as it passes throyugh the rule operator..
- **BookClickFilter** provides a more complex set of parameters that use both a Java XOM and hierarchical xml parameters. This requires mapping of the tuple fields into hierachical object models. It also shows how different output parameters from the rules can be fed to different streams.

These two patterns share a common information model but use different mapping techniques between the streams representations and the way the information is referenced in the rules.

The Classification pattern uses a very simple mapping approach passing each attribute over a single simple type (or list of simple types). This style of mapping can be supported by the Rule Executor Operator without the need for any customization code. Any mapping to structured business objects needs to be performed in the ruleset as part of the operation ruleflow.

The Filtering pattern extends this approach to perform the mapping to Java classes in the Ruleset Executor Operator using custom plugin code. This allows more complex stream structures to be supported and mapped directly to executable object models within ODM avoiding serialization and delivering higher performance. The filtering pattern also shows how to support the use of schema based eXecution Object Models and mapping different return parameters to different streams.

Information Model Overview

This section describes the structure of the information being processed in the solution.

The streams are processing customer clicks on a web site or application which contain information about the click that they take and the state of their order.

Each field is tabulated below to describe its purpose in the scenario.

Table 2.1 BookClick input attributes

Attribute	ODM type	Description
customerID	String	Identifier of the customer
clickAction	String (ClickActionType)	Action taken by this click VIEW, ADD, REMOVE, ORDER
clickTimestamp	Date	Timestamp of the click
isbn	String	Book ISBN
title	String	Book title
author	String	Book author
price	BigDecimal (in Class Book)	Book list price on website
basketValue	BigDecimal	Value of shopping basket
items	List<String>	List of book isbn numbers in basket
book	Book	Book object passed as a Java class

The streams application then passes this information to the Ruleset Executor Operator which then invokes an ODM Ruleset which then generates a response stream.

This response stream extends the incoming stream with two new attributes provided from the rules that describe the classification and rationale as shown in table 2.2.

Table 2.2 BookClick response attributes

Attribute	ODM Type	Description
basketValue	BigDecimal	Value of shopping basket
items	Vector<String>	List of book isbn numbers in basket
classification	String (BookClassification)	Classification provided by the rules: BROWSING, FILLINGBASKET, VALUEBASKET, OFFERBASKET, PLACEORDER
rationale	String	Textual rationale for the classification
bookEvent	XML BookEvent Object	XML representation of filtered event

The information required for the decisions is represented in ODM through variables that can either be simple types (as shown in the tables) or complex types represented in a Business Object Model. The next section describes how this can be represented in ODM.

Simple (unstructured) ODM Information Models

When using simple flat information models, the book click fields are held in ODM variable sets and can be mapped to and from the parameters passed into the ruleset at execution time. A simple variable set containing flat unstructured data is show below in figure 2.2.

Figure 2.2 Simple (Non-structured) Variables

Variable Set: SimpleVariables

Name	Type	Verbalization	Initial Value
isbn	java.lang.String		
title	java.lang.String		
author	java.lang.String		
price	java.math.BigDecimal		
classification	BookClassification	the classification	BookClassification.BROWSING
rationale	java.lang.String	the rationale	""
basketValue	java.math.BigDecimal	the value of the basket	
items	java.util.List	the books in the basket	
customerID	java.lang.String	the customer ID	
clickTimestamp	java.util.Date	the time of the click	
clickAction	ClickActionType	the click action	

When using simple variables the streams ODM Ruleset Executor operator can provide automatic mapping to the attributes in the streams. The **Name** field has to correspond to name of the attribute in the stream and automatic mapping will be performed by the Ruleset Operator using the corresponding **Type** mapping.

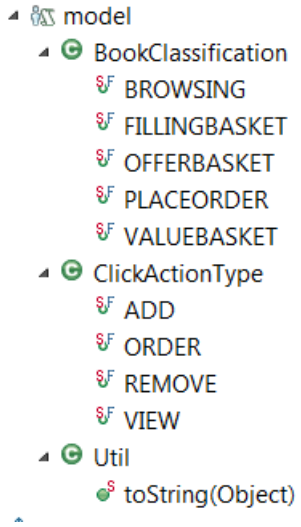
The verbalization defines how the variables will appear to the rules. Note that the isbn, title, author and price variables do not have a verbalization and are not directly accessible from the rules. These variables (when present) will be used to construct a book object as described later.

For each variable ensure that the name matches the attibute name used in streams. This means that the variable can be mapped automatically. You should also ensure that the Type corresponds to the mapped typed used in streams. See the Data type conversions between ODM and InfoSphere Streams section in the InfoSphere streams knowledge center for a more detailed description of the conversions provided by the ODM Ruleset Executor operator.

Any variables that are going to be provided back to the streams should also be initialized. Failure to do this may result in null pointer exceptions when the rules execute. In this case the classification and rationale fields are initialized. You may also like to create and initialize any local variables that ease the decision making flow and design.

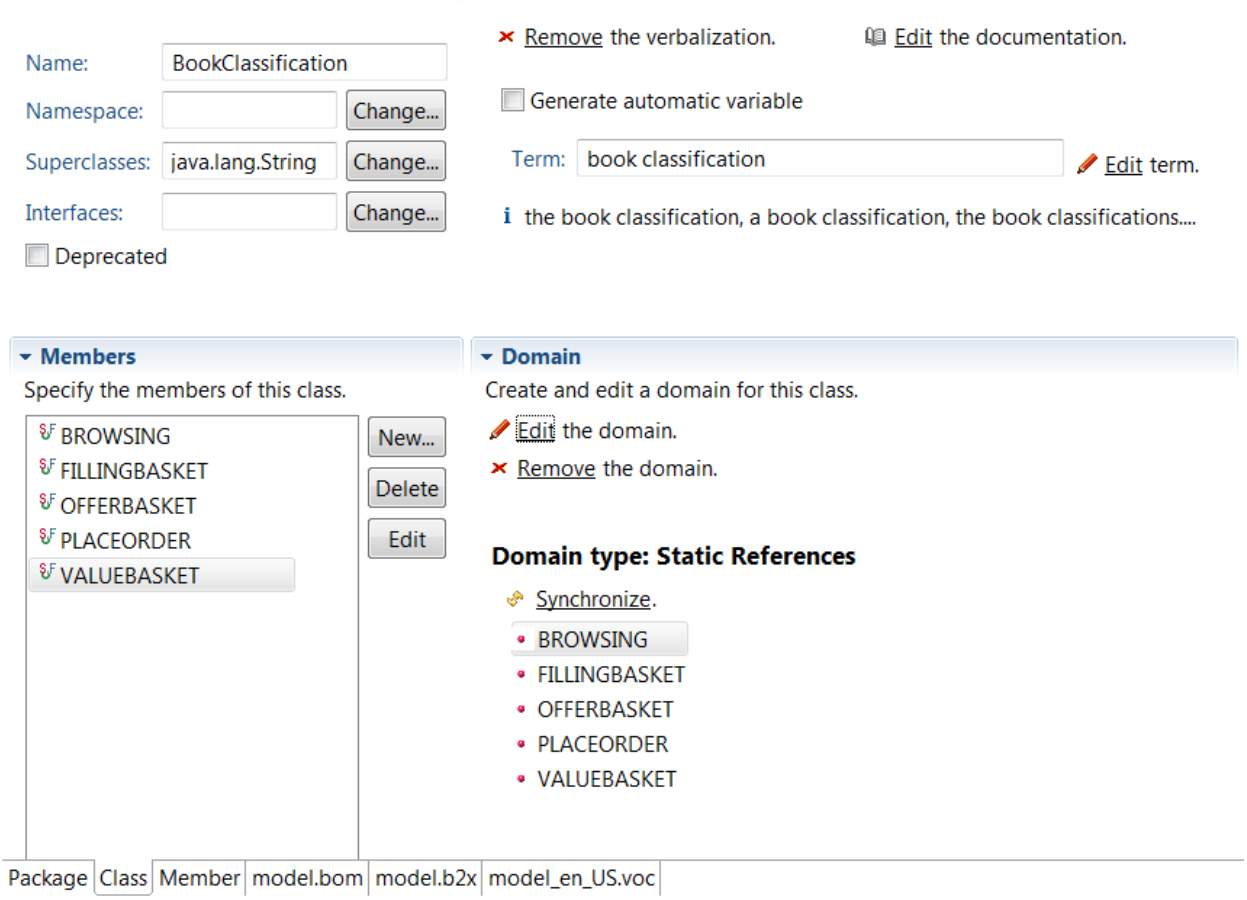
Note also that two of these variables (classification and clickAction) represent domains or enumerated values and use specific classes (BookClassification and ClickActionType) to define the possible values that the underlying String can take. This means that the rules can be constrained to only permit the enumerated values to be used when composing rules that reference variables typed in this way. This is configured as shown in figure 2.3 below.

Figure 2.3 Simple Parameter Business Object Model



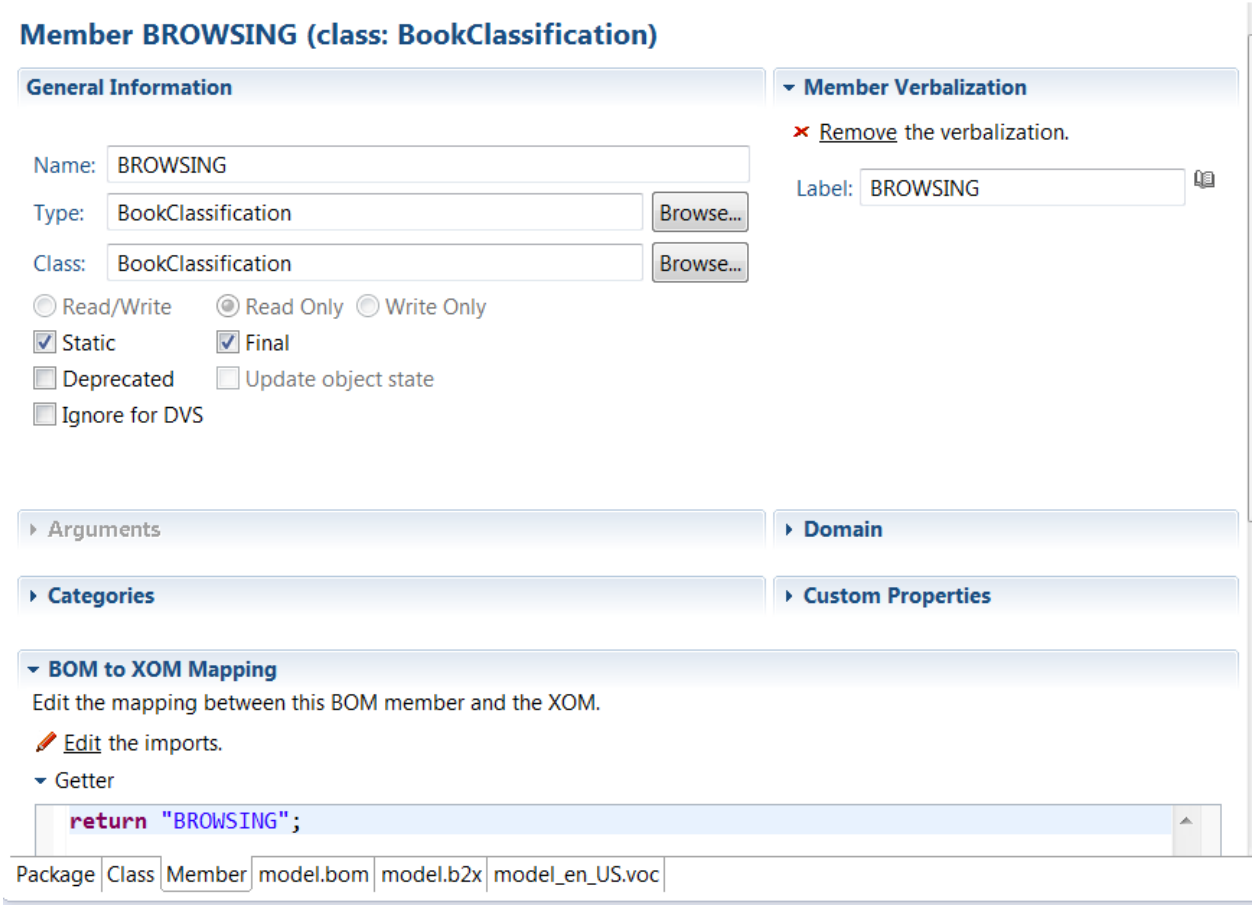
This BOM declares two domains that can be used to constrain the values used in the rules when selecting options for clickActions or classifications. Each domain is specified as a class extending a java.lang.String and providing static attributes for each enumeration as shown in figure 2.4.

Figure 2.4 Configuring a Domain for enumerated values



Each domain value need to have an implementation that verbalizes it and provides a retrn value as shown in figure 2.5 below.

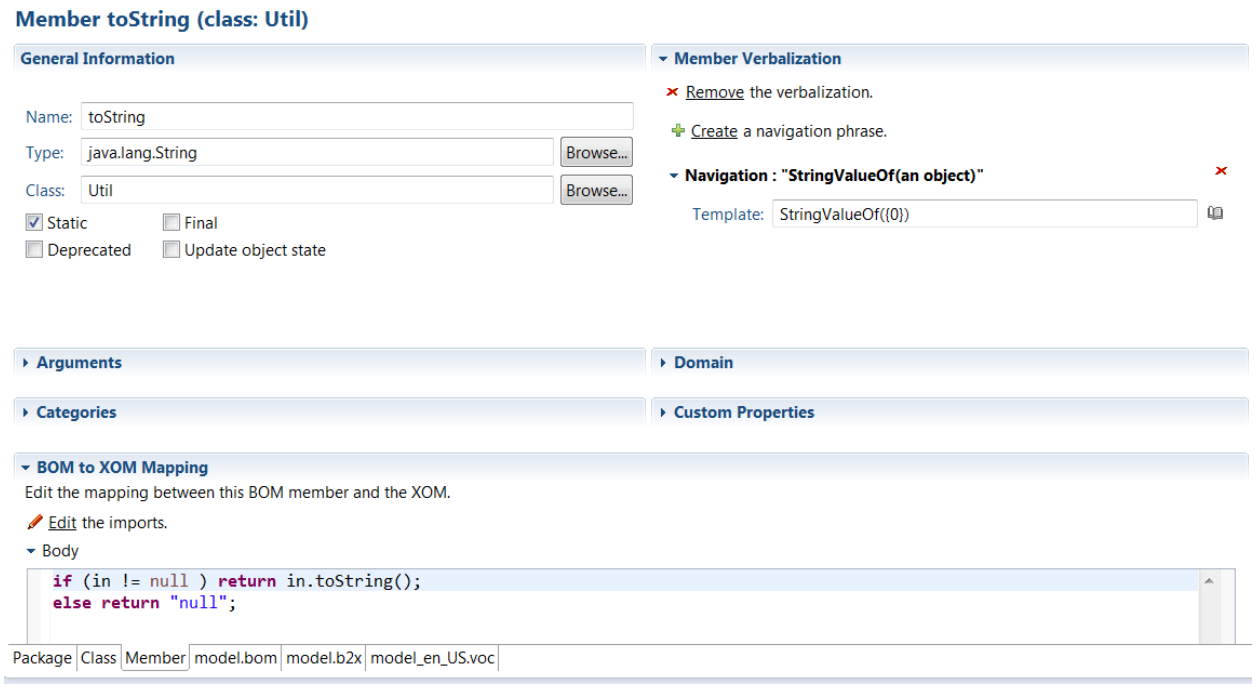
Figure 2.5 Defining Domain Members



This approach forces the rules to reference only these explicit values if the string is typed according to a given domain. When the values used for these domains are very dynamic and are held in another system (or spreadsheet) it is possible to write integration code that can synchronize with that source and thus maintain a consistent set of enumerated values. See the Working with Domains section in the IBM Operational Decision Manager Knowledge Center for details.

The Business Object model also includes a **Util** class with a verbalized operation which allows any object to be expressed as a string using the verbalization: **StringValueOf(an object)** . This allows, for example, numbers and dates to be included in a rationale.

Figure 2.6 Providing Utility operations in the vocabulary



Structured ODM Information Models

The information in ODM can also be managed through variables with complex types represented in a Business Object Model. These variables are also held in ODM variable sets and can be mapped to and from the parameters passed into the ruleset at execution time. The structured variable set containing structured data variables is show below in figure 2.7.

Figure 2.7 Structured Variables

Variable Set: StructuredVariables

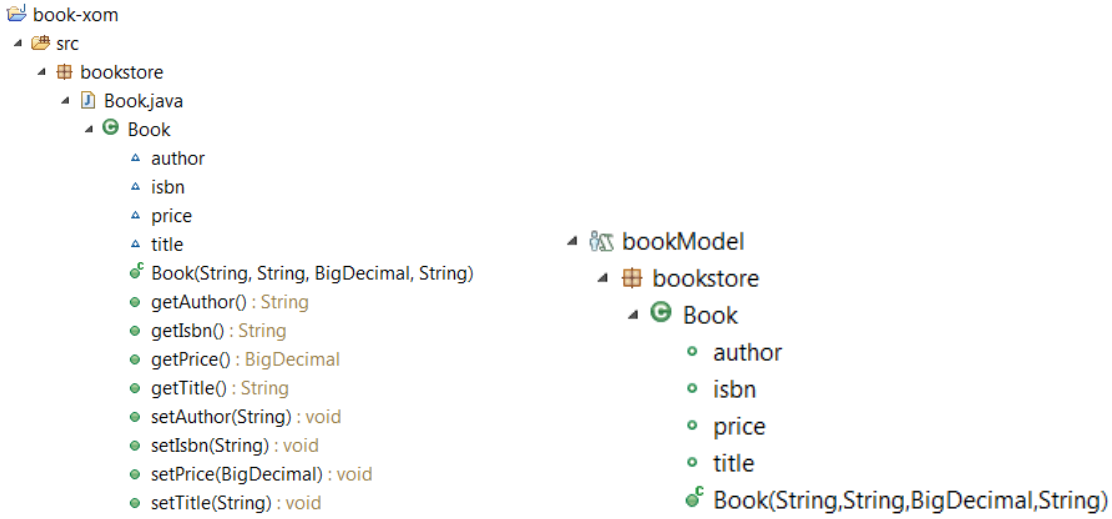
Name	Type	Verbalization	Initial Value
book	bookstore.Book	the book	
bookEvent	bookevent.BookEvent	the book event	

These variables are passed to the streams using a complex type which in the case of the book (bookstor.Book) is based on a Java XOM while the bookEvent (bookevent.BookEvent) is based on an xml schema and will be passed to streams as an xml string.

In this example we have the concept of a book which has four attributes and is implemented as a Java XOM to provide a structured type for the input and output

parameters. The BOM is shown below in figure 2.8 in Rule Designer showing the details of the class and the members. This has been based on the Java class illustrated alongside.

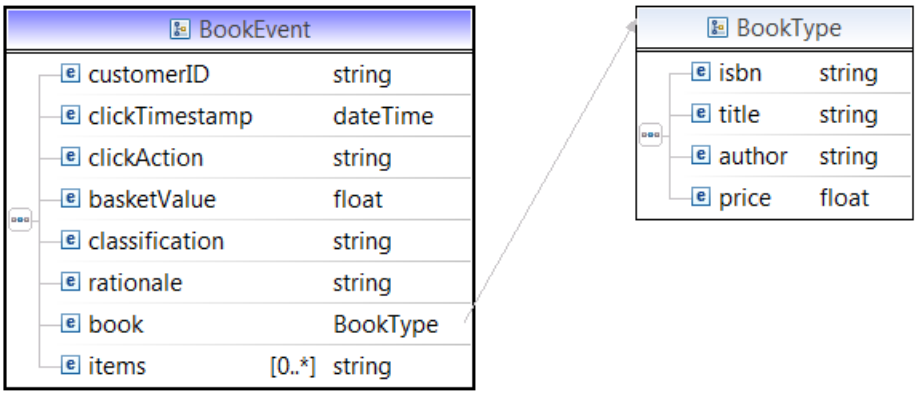
Figure 2.8 Book Java XOM and BOM



This allows book objects to be represented as Java objects in the streams mapping and interpreted directly in the rule engine without any serialization in ODM.

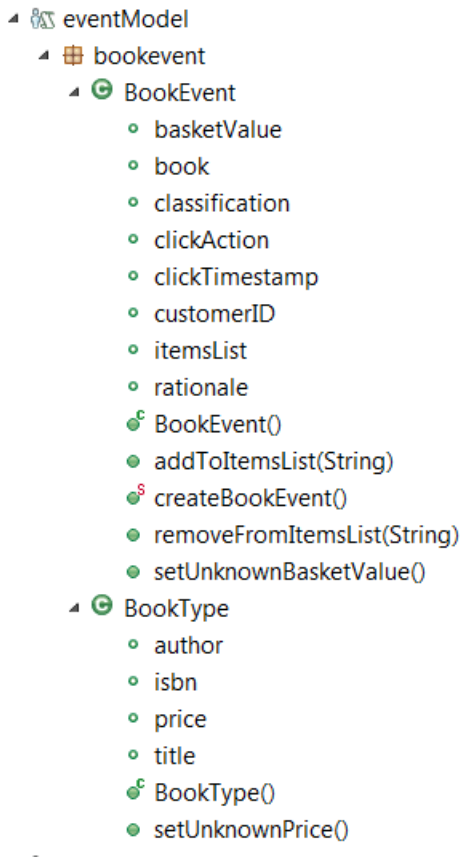
The BookClickFilter ruleset also uses a hierarchical model to represent a BookEvent. This is based on XML schema and is added as an output parameter so that a book event can be generated by the rules when further action processing or exceptions occur. All the information from the input parameters and the output classifications are included in the book event data. The XML schema is shown below in Figure 2.9.

Figure 2.9 Book Event Schema



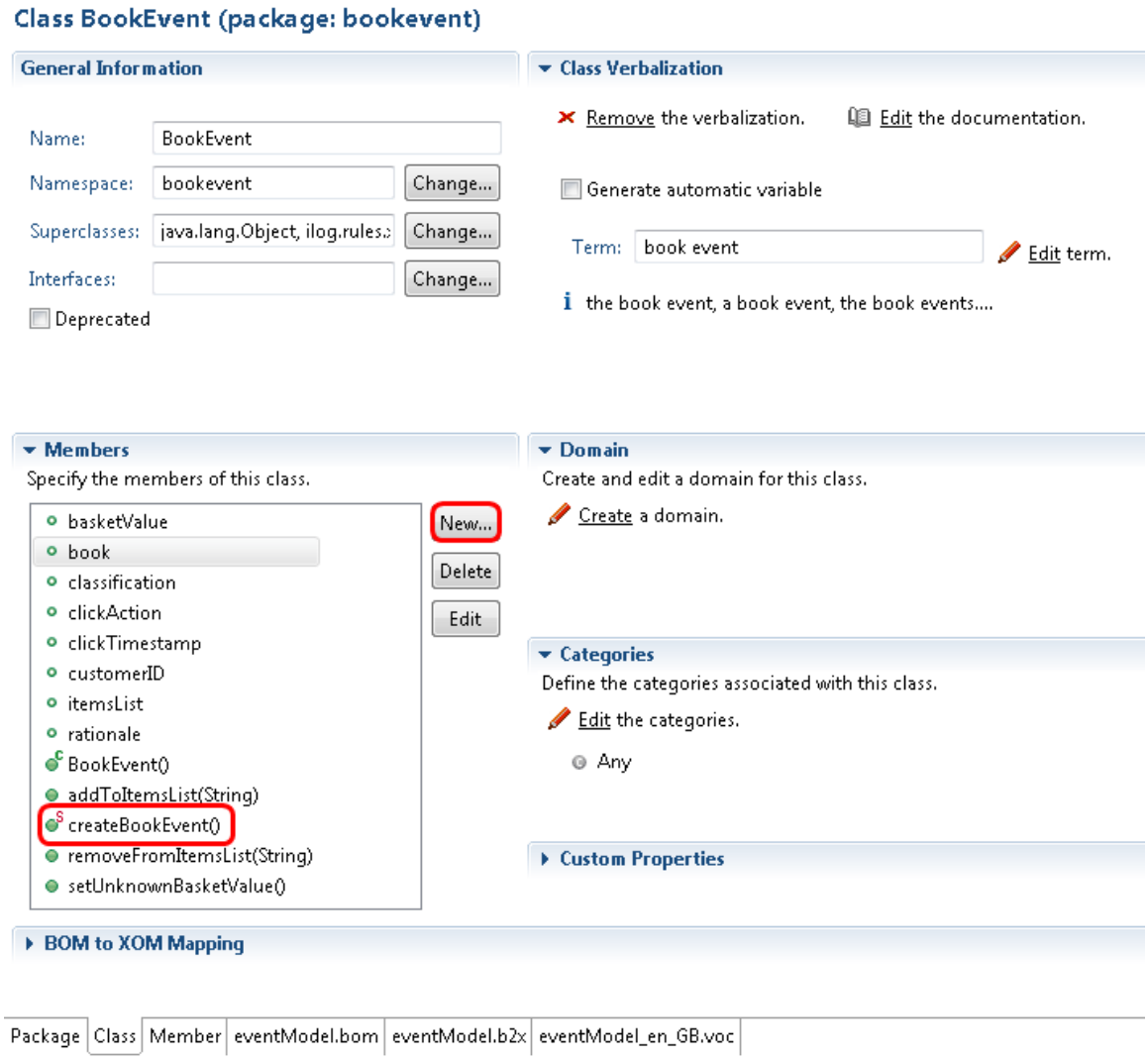
The BookEvent BOM can be generated automatically from the XML schema in Rule Designer as shown in figure 2.10 below.

Figure 2.10 BookEvent BOM



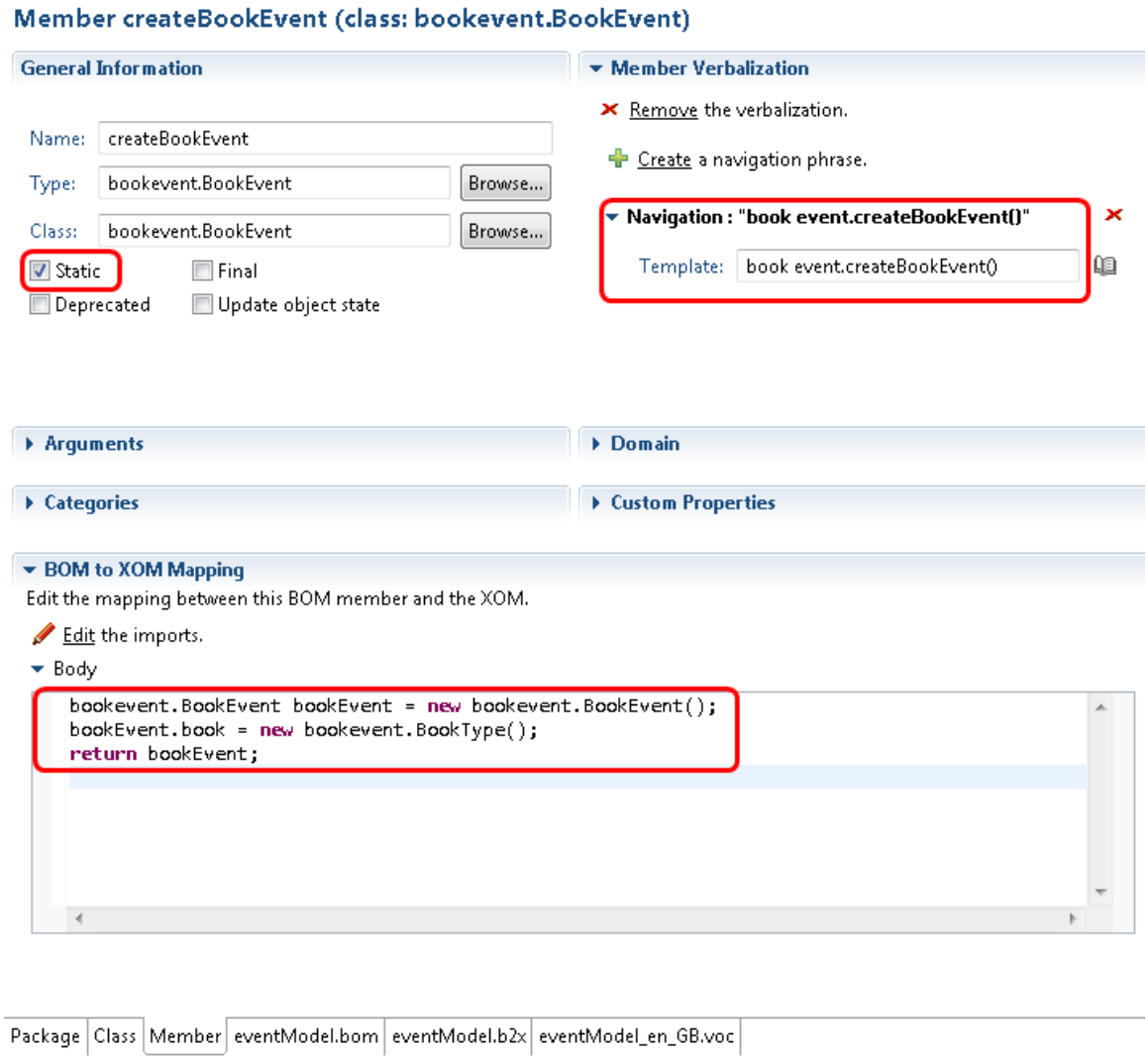
In this BOM methods have been added to manipulate the items list (addToItemsList, removeFromItemsList) and we also need to define a method to create a new BookEvent from the rules. This new member createBookEvent() is added as a static method on the BookEvent class so that book events can be created in the rules for those book click data that requires further action. See Figure 2.11 below on using the New button to create a createBookEvent() member in Rule Designer.

Figure 2.11 Adding operations to a BOM



The new createBookEvent() should be a static method and a BOM to XOM Mapping needs to be defined which implements the behaviour and instantiates the BookEvent class hierarchy including the BookType member. The method also needs to be verbalized in the member verbalization to allow access from rules. The BOM to XOM mapping in the Rule Designer is shown in Figure 2.12 below in the BOM Member page for createBookEvent().

Figure 2.12 Defining operation behaviour using BOM to XOM Mapping

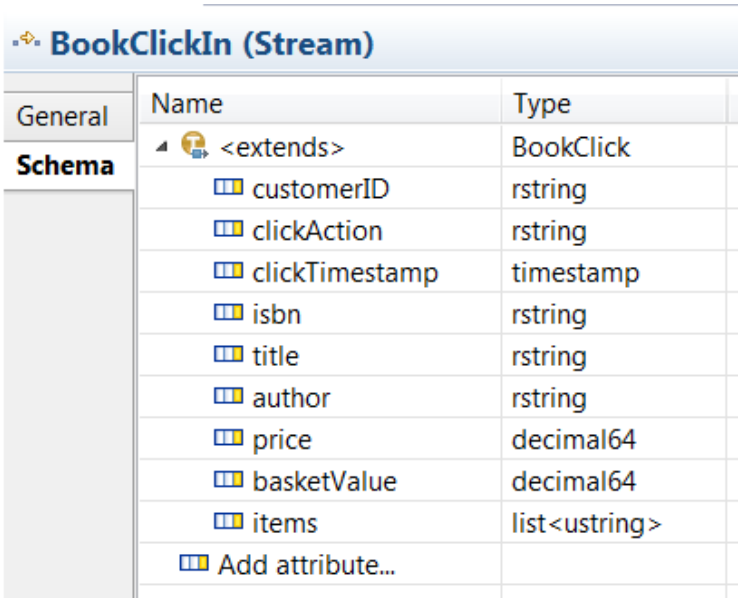


This now completes the information models needed by both patterns.

Streams Information models

This section describes the structure of the streams and how they represent the information being processed in the solution. Figure 2.13 describes the structure of the BookClick input stream which corresponds to the book click information coming from the customer channels.

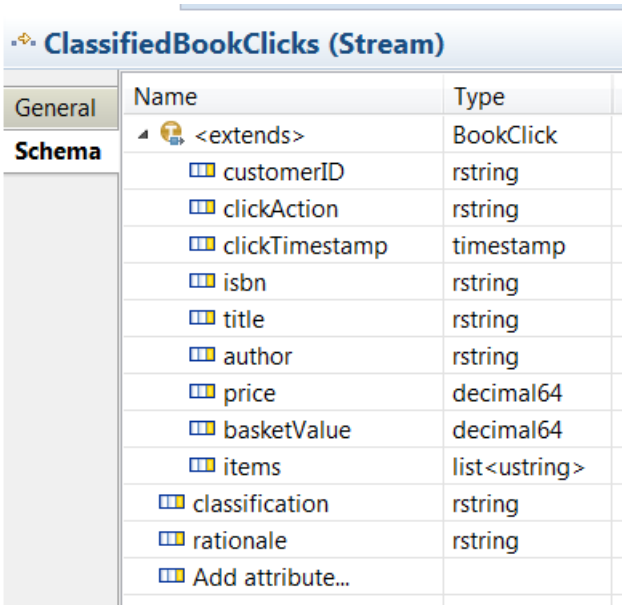
Figure 2.13 BookClickIn Stream information model



The purpose of each field has been already described in Table 2.1.

The streams application the passes this information to the Ruleset Executor Operator which then invokes an ODM Ruleset and then maps the returned parameters to a ClassifiedBookClicks output stream as shown in figure 2.14.

Figure 2.14 ClassifiedBookClicks Stream information model



This stream simply extends the incoming stream with two new attributes provided from the rules that describe the classification and rationale.

In these models, rstring is mostly used for String values except when used in a list. In this case ustring is used. There are limitations with using list<rstring> when mapping to ODM parameters as the items in the list cannot be accessed in ODM rules. Using list<ustring> removes the limitation.

The difference between ustring and rstring are summarized in Table 2.3 below:

Table 2.3 Representation and Java object type for rstring and ustring

SPL type	Representation	Java object type
ustring	String of UTF-16 Unicode characters, based on ICU library	java.lang.String
rstring	Sequence of raw bytes that supports string processing when the character encoding is known	`com.ibm.streams.operator.types.RString`

For more details of SPL types, see Types and Working with SPL types from the InfoSphere Streams Knowledge Center.

Classification Pattern

The Classification pattern describes how an ODM JSE Rule integration server (RES) can be embedded into a streams operator allowing the stream processing to be configured dynamically by rules. In this case the stream contains the individual customer actions when building up a book order including viewing, adding or removing a book from the order. The rules can then classify each tuple in the stream which can then be used in later streams analytics processing.

Classification Pattern Streams Application

In this pattern a stream of tuples is passed through the Rules Executor operator and the attributes modified by the rules. The attributes of tuples in the streams are passed individually as parameters to the rules. Each parameter is a simple primitive type (or list of primitive types) allowing the rules to refer to each attribute in the stream individually. Rules and decision tables can then be written in an ODM ruleset to classify each tuple by assigning values to particular output parameters which will then be passed onto the output stream.

A simple implementation of ths pattern is shown below.

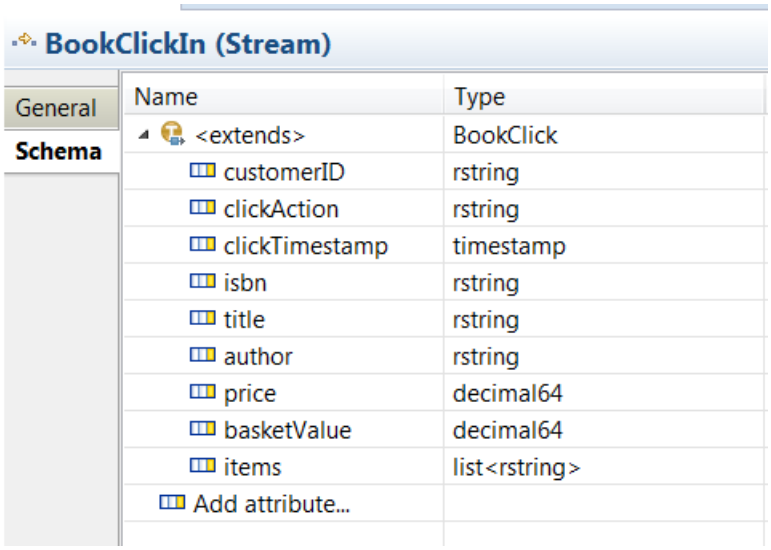
Figure 3.1. Classification Pattern application

The three operators are now described in the following sections.

CSVIn FileSource operator

The input to the Rules Operator would come from the Systems of Engagement and reflect various actions taken by customers when browsing the Book Order sites. This is emulated in this sample by the CSVIn FileSource operator which reads tuples in from a file (BookClicksIn.txt) using a csv format and outputs them on the BookClickIn stream as shown in figure 3.2 below.

Figure 3.2 BookClickIn Output stream schema

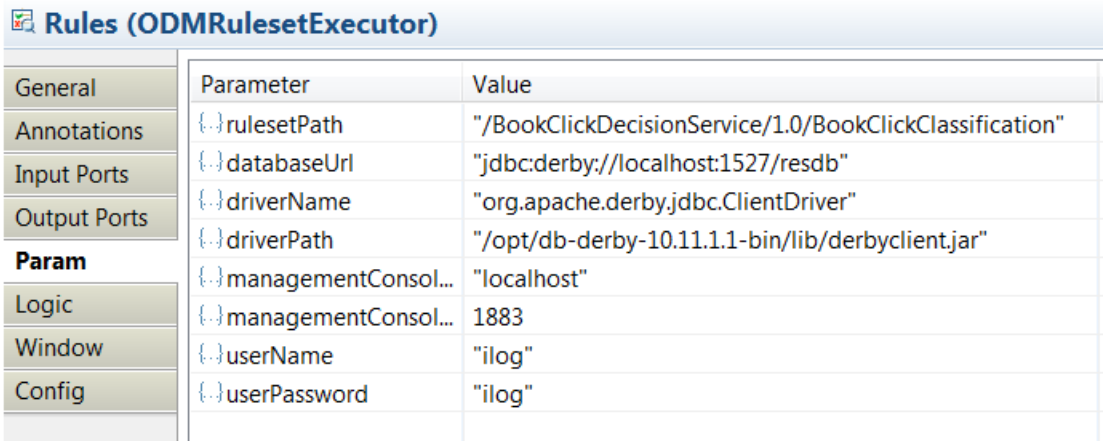


This stream uses the BookClick type to define the stream attributes. In this example the attributes are mapped into simple type parameters in ODM.

Rules ODMRulesetExecutor Operator

This stream is passed to the Rules (ODMRulesetExecutor) operator which needs to be configured to load the correct ruleset from the Decision Server repository and also to respond automatically to updates as described in the installation and configuration annex and summarized in figure 3.3 below.

Figure 3.3 Rules (ODMRulesetExecutor) configuration parameters



This configuration uses the latest deployed version of the BookClickClassification ruleset with interfaces defined by the decision service version BookClickDecisionService/1.0.

The remaining parameters define the connection to the Rule Execution Server database (based on derby in this example) and the Rule Execution Server console. This allows ruleset updates to be deployed dynamically from ODM Rule Designer or Decision Center without needing to stop or redeploy the streams application.

The ODMRulesetExecutor provides a default mapping from the stream tuple to ruleset parameters for all primitive attribute types. List of primitive attribute types are also supported but care must be taken as the internal types are not correctly mapped.

Two known limitations are:

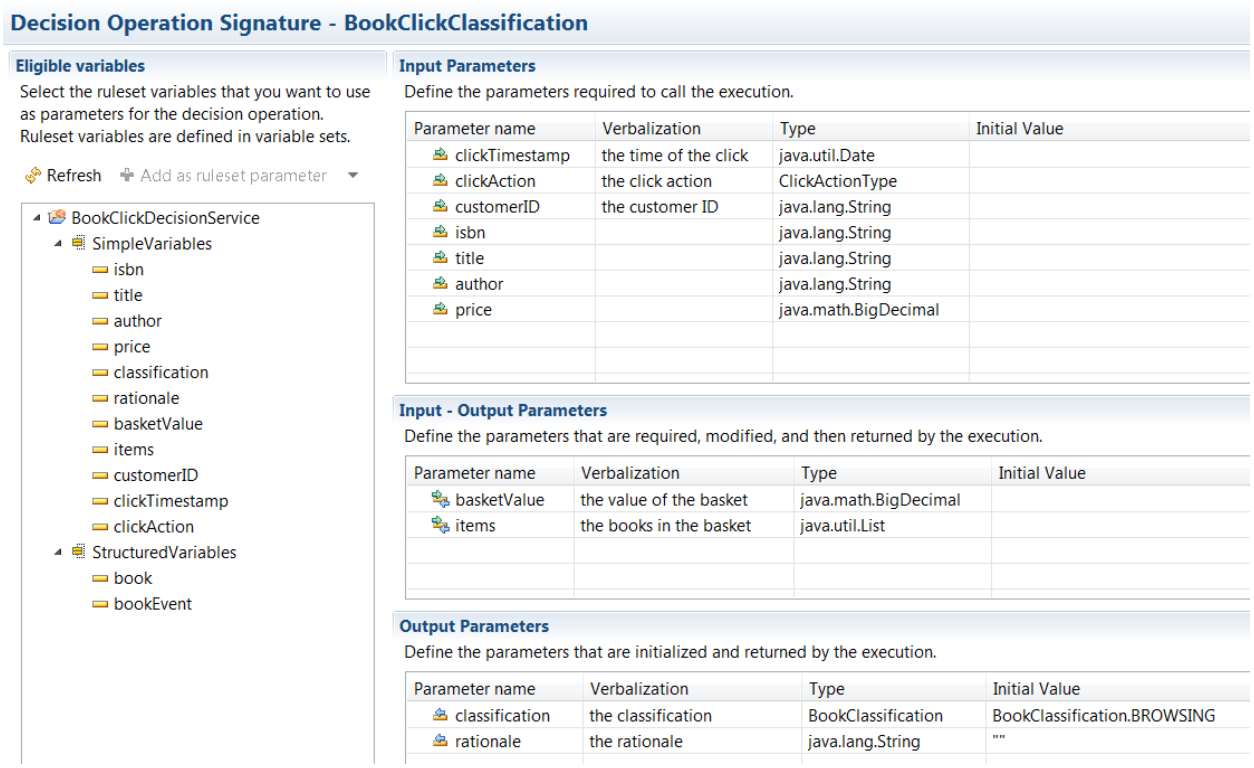
- Timestamp primitive types are mapped as java.sql.Date by default which is not fully supported in ODM.
- list<Type> maps to a java.util.List<streams Type> rather than the mapping supported for a primitive type (e.g list<rstring> instead of list<java.lang.String> - this can be overcome by using ustring instead of rstring as the streams type).

To overcome these limitations you need to customize the mapping as described in the Filtering pattern.

BookClickClassification Ruleset

The BookClickClassification ruleset is defined as an operation within the BookClickDecisionService. To complete the integration with streams we need to define the decision operation signature that will be mapped to streams attributes. This replaces the ruleset parameter definitions used in classic rule projects in previous versions of ODM.

Figure 3.4 BookClickClassification Decision Operation signature

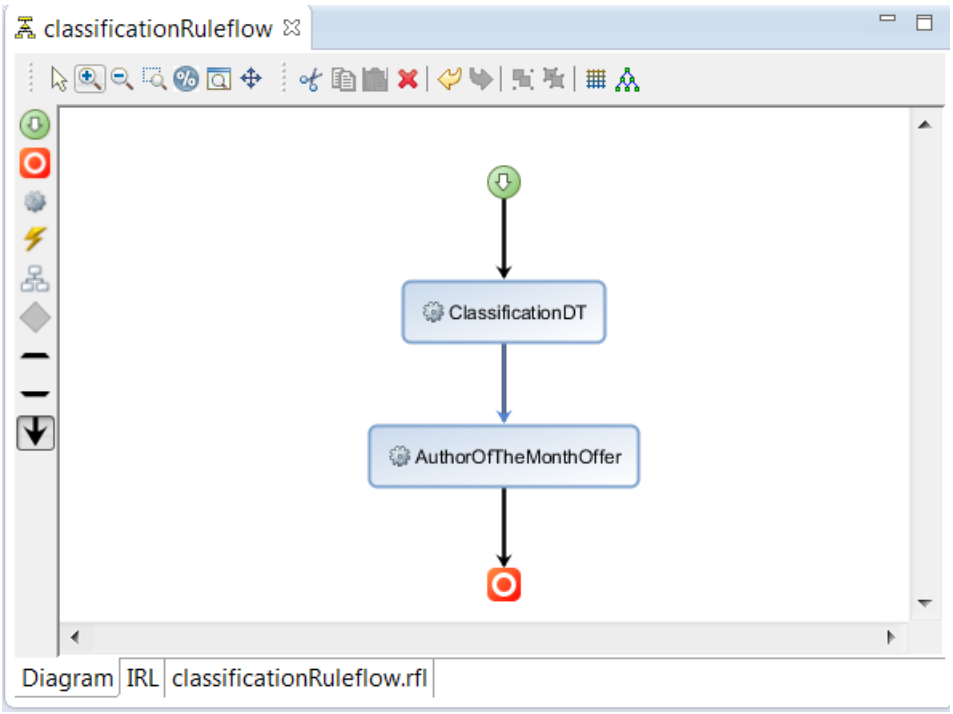


Ths shows which variables will be used as input and which as output. Note that these are all drawn from the SimpleVariables variableset as this ruleset and pattern will not use hierarchical parameters.

Note that the book parameters (isbn,title,author and price) are not verbalized as these will be mapped into an object as the first step in the ruleflow. This allows the rules to be developed and shared against a fixed set of variables.

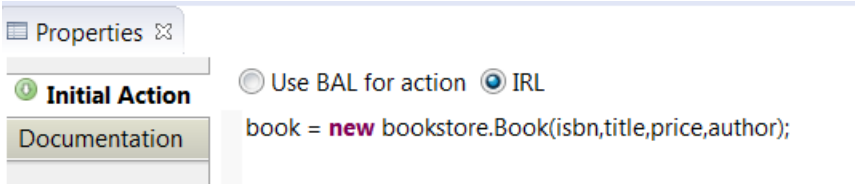
Once the signature is defined, the rules and ruleflow can be developed. This article does not attempt to provide a detailed description of ruleset design practices so a very simple ruleflow is used to calculate the classification and rationale as shown in figure 3.5.

Figure 3.5 Classification Ruleflow



The ruleflow first creates a book object in the initial task as shown in figure 3.6, then invokes the ClassificationDT which uses the clickAction with the basketValue to determine a base classification and rationale as shown in figure 3.7. Finally the flow evaluates whether to apply an offer to this potential order as shown in figure 3.8.

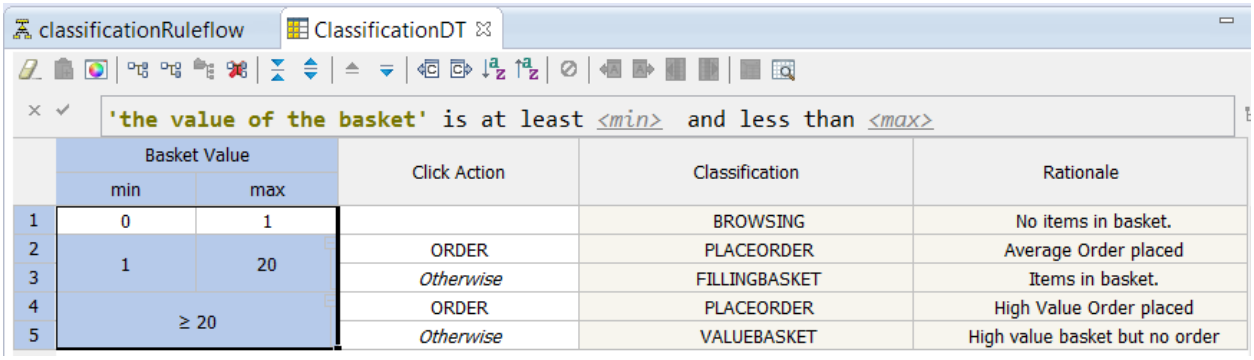
Figure 3.6 Initial Action



Each task in a ruleflow has the ability to specify an initial action using rules. In this case this is the entry task so will occur at the beginning of the ruleflow. The rules can be written in the BAL language (as used by business users using the verbalized phrases) or as IRL which is a java like language allowing access to all variables whether they are verbalized or not. In this case the IRL creates a new Book object from the hidden parameters and assigns it to the visible book variable allowing the values to be referred to in the rules. (See figure 3.8).

After the variables are initialized the Classification decision table is invoked.

Figure 3.7 ClassificationDT Decision Table

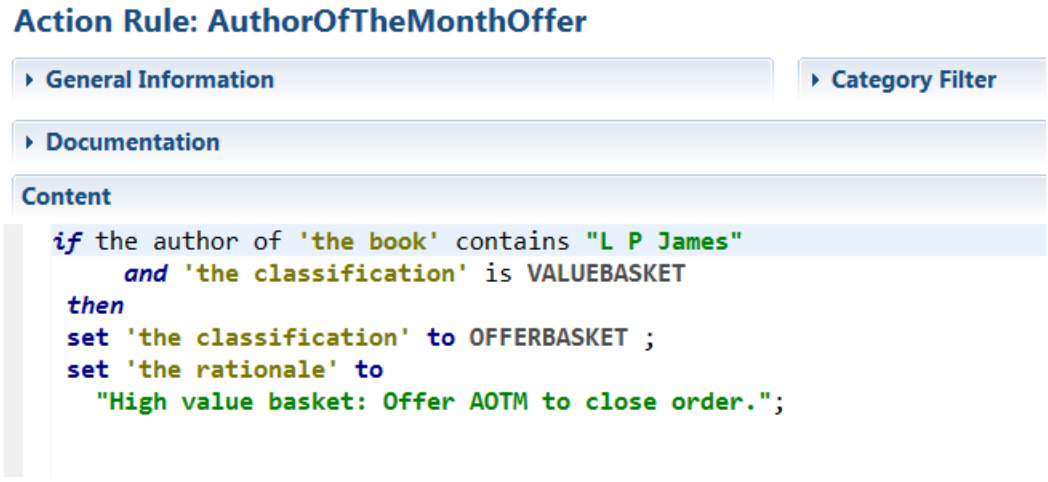


	Basket Value		Click Action	Classification	Rationale
	min	max			
1	0	1		BROWSING	No items in basket.
2	1	20	ORDER	PLACEORDER	Average Order placed
3			*Otherwise*	FILLINGBASKET	Items in basket.
4	≥ 20		ORDER	PLACEORDER	High Value Order placed
5			*Otherwise*	VALUEBASKET	High value basket but no order

This provides a classification and rationale for a range of combinations of basket value and click action.

After the base classification has been determined, the AuthorOfTheMonthOffer rule checks to see if the customer is looking at a book that has the author of the month offer as shown in figure 3.8.

Figure 3.8 AuthorOfTheMonthOffer rule

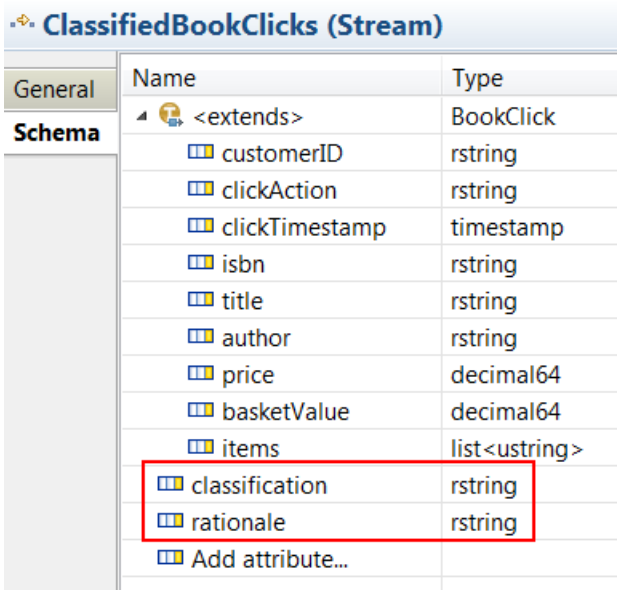


Note that in this case the rule refers to "**the author of 'the book'**" rather that referring to **the author** variable directly as this is not visible and will not have a value when the book is passed as an object directly from streams. This allows the same rules to be used in different operations with different signatures.

While this is a simple example it shows how the rule engine can process the tuple attributes and classify them according to the rules defined by the business.

After the rules have been processed the results are passed to the ClassifiedBookClicks stream as shown in figure 3.9 below.

Figure 3.9 ClassifiedBookClicks Output stream schema



This stream duplicates the original BookClickIn type but adds the two new attributes whose values are calculated by the rules:

- Classification – indicates the significance of the click in the book order processing. Typical values expected from the rules include UNKNOWN, BROWSING, FILLINGBASKET, VALUEBASKET and various eligible discounts
- Rationale – provides a free text field to say why the particular classification has been applied.

The ClassifiedBookClicks stream can then be sent on to other operators in the streams application or in the case of this sample output to the CSVout FileSink operator allowing the results of a job to be observed in the classifiedBookClicksOut.txt file.

Classification Pattern Summary and Execution

The complete listing for the classificationPattern application is provide in listing 3.1 below.

Listing 3.1 ClassificationPattern application SPL listing

```
namespace application ;
use com.ibm.streams.rules.odm::ODMRulesetExecutor ;
composite ClassificationPattern()
{
  type
    BookClick = rstring customerID, rstring clickAction,
      timestamp clickTimestamp, rstring isbn, rstring title, rstring author,
      decimal64 price, decimal64 basketValue, list<ustring> items ;
    ClassifiedBookClick = rstring customerID, rstring clickAction,
      timestamp clickTimestamp, rstring isbn, rstring title, rstring author,
```

```
        decimal64 price, decimal64 basketValue, list<ustring> items,
        rstring classification, rstring rationale ;

    graph
        (stream<BookClick> BookClickIn) as CSVIn = FileSource()
        {
              param
                    file : "bookClicksIn.txt" ;
                    format : csv ;
        }

        () as CSVOut = FileSink(ClassifiedBookClicks)
        {
              param
                    file : "classifiedBookClicksOut.txt" ;
                    format : csv ;
        }

         (stream<BookClick, tuple<rstring classification, rstring rationale>>
              ClassifiedBookClicks) as Rules = ODMRulesetExecutor(BookClickIn)
        {
              param
                    rulesetPath :
                     "/BookClickDecisionService/1.0/BookClickClassification" ;
                    databaseUrl : "jdbc:derby://localhost:1527/resdb" ;
                    driverName : "org.apache.derby.jdbc.ClientDriver" ;
                    driverPath :
                     "/opt/db-derby-10.11.1.1-bin/lib/derbyclient.jar" ;
                    managementConsoleHost : "localhost" ;
                    managementConsolePort : 1883 ;
                    userName : "ilog" ;
                    userPassword : "ilog" ;
                    managedXomDeployedOnDb : true ;
        }
}
```

The classification pattern sample may be executed in a standalone stream to demonstrate the integration techniques. The use of CSV files for the input source and sink make this an easy option for investigating the rules.

A sample input file has been established in bookClicksIn.txt as shown in listing 3.2.

Listing 3.2 bookClicksIn.txt Sample input stream in csv format

```
#rstring customerID, rstring clickAction,timestamp clickTimestamp,
#rstring isbn, rstring title, rstring author, decimal64 price,
#decimal64 basketValue, list<rstring> items
"A-111","ADD",(5000000, 0, 0),"S-111","Night","G Jones",7.99,7.99,["S-111"]
"A-111","ADD",(5000001, 0, 0),"S-222","Quiet Day","L P James",9.99,17.98,["S-111","S-222"]
"A-111","VIEW",(5000002, 0, 0),"S-333","Sun in the Sky","L P James",8.99,26.97,["S-111","S-222","S-333"]
"B-222","ADD",(5000003, 0, 0),"S-444","Short Stories","V Hurst",4.99,4.99,["S-444"]
"B-222","REMOVE",(5000004, 0, 0),"S-444","Short Stories","V Hurst",4.99,0.0,[]
"B-222","ADD",(5000005, 0, 0),"S-555","Day","H R Smith",6.99,6.99,["S-555"]
"B-222","ORDER",(5000006, 0, 0),"S-666","My Life","F Bloggs",5.49,12.48,["S-555","S-666"]
"E-555","VIEW",(5000006, 0, 0),"S-777","Go Like the Wind","S Speed",12.99,0.0,[]
"E-555","ADD",(5000007, 0, 0),"S-888","Long Stories","L P James",15.99,0.0,["S-888"]
"E-555","VIEW",(5000008, 0, 0),"S-999","Hobsons Choice","H R Smith",2.99,15.99,["S-888"]
"D-444","VIEW",(5000009, 0, 0),"S-999","Hobsons Choice","H R Smith",2.99,0.0,[]
```

When the application is executed the result can be seen in the classifiedBookClicksOut file. The two parameters output from the rules (classification and rationale) have been moved to the next line for readability.

Listing 3.3 classifiedBookClicksOut.txt Sample output stream in csv format

```
"A-111","ADD",(5000000,0,0),"S-111","Night","G Jones",7.99,7.99,["S-111"],
         "FILLINGBASKET","Items in basket."
"A-111","ADD",(5000001,0,0),"S-222","Quiet Day","L P James",9.99,17.98,["S-111","S-222"],
         "FILLINGBASKET","Items in basket."
"A-111","VIEW",(5000002,0,0),"S-333","Sun in the Sky","L P James",8.99,26.97,["S-111","S-222","S-333"],
         "OFFERBASKET","High value basket: Offer AOTM to close order."
"B-222","ADD",(5000003,0,0),"S-444","Short Stories","V Hurst",4.99,4.99,["S-444"],
         "FILLINGBASKET","Items in basket."
"B-222","REMOVE",(5000004,0,0),"S-444","Short Stories","V Hurst",4.99,0,[],
         "BROWSING","No items in basket."
"B-222","ADD",(5000005,0,0),"S-555","Day","H R Smith",6.99,6.99,["S-555"],
         "FILLINGBASKET","Items in basket."
"B-222","ORDER",(5000006,0,0),"S-666","My Life","F Bloggs",5.49,12.48,["S-555","S-666"],
         "PLACEORDER","Average Order placed"
"E-555","VIEW",(5000006,0,0),"S-777","Go Like the Wind","S Speed",12.99,0,[],
         "BROWSING","No items in basket."
"E-555","ADD",(5000007,0,0),"S-888","Long Stories","L P James",15.99,0,["S-888"],
         "BROWSING","No items in basket."
"E-555","VIEW",(5000008,0,0),"S-999","Hobsons Choice","H R Smith",2.99,15.99,["S-888"],
         "FILLINGBASKET","Items in basket."
"D-444","VIEW",(5000009,0,0),"S-999","Hobsons Choice","H R Smith",2.99,0,[],
         "BROWSING","No items in basket."
```

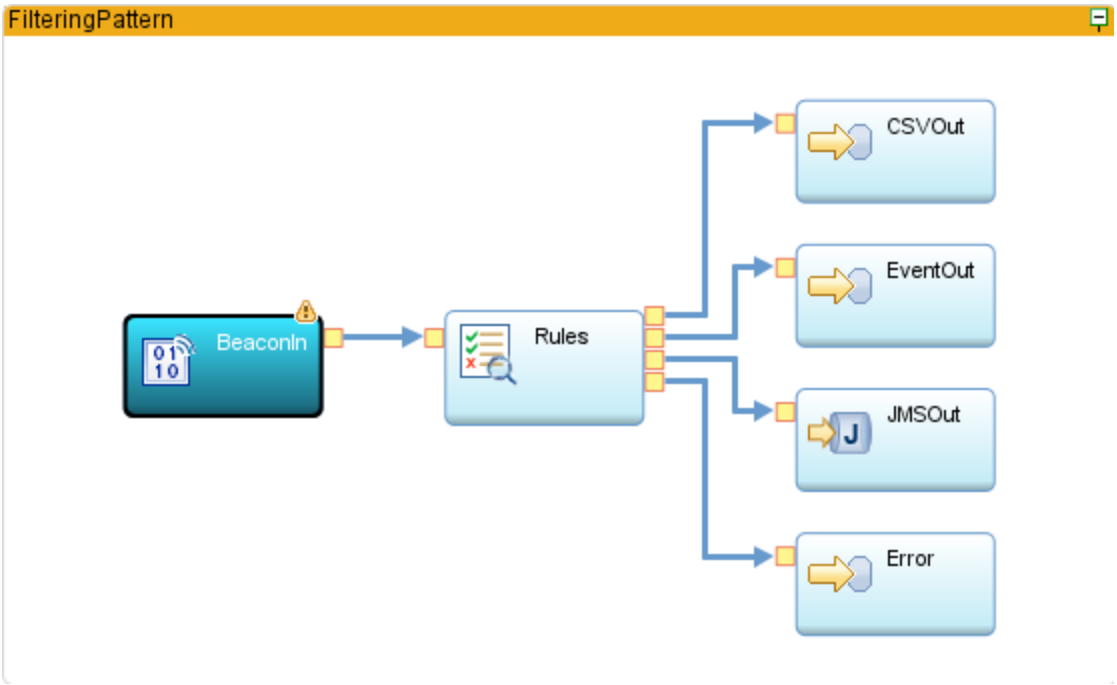
Filtering Pattern

In this pattern rules are used to identify particular tuples that are of significance to later analytics processing and remove those that are not relevant. This filtering may apply to the main output stream but in this case an additional stream is generated that identifies significant events which can then be routed to operational decision making systems to apply suitable action responses.

Filtering Pattern Streams Application

This pattern builds on the Classification Pattern described earlier using the same input stream structure. In this case the pattern shows how to map the tuple attributes into java objects that can be passed as parameters to the ODM Ruleset executed by the ODMRulesetExecutor operator. The response parameters (containing these objects) are then mapped to a number of ports and this example shows how the parameters are mapped and formatted to meet the requirements of a number of streams as shown in figure 4.1 below.

Figure 4.1 Filtering Pattern application



The input stream (**BookClicksIn**) is provided through a Beacon operator (**BeaconIn**) that generates a sequence of configurable BookClick tuples with random book **price** and **basketValue** attributes.

The ODMRulesetExecutor operator (**Rules**) invokes some simple rules to first classify each tuple (as described for the Classification pattern) and then creates an optional event output parameter for tuples that are classified **OFFERBASKET**.

The default output port provides a classified stream (**FilteredBookClicksOut**) with a similar structure as the **BookClicksIn** stream but including the classification and rationale fields. This is sent to the FileSink operator (**CSVOut**) which allows the classified stream tuples to be written to a file (filteredBookClicksOut.txt) for later assessment.

The customized mapping in the **Rules** operator then takes the "bookEvent" parameter (if present) and maps it into two ports generating streams with different formats:

- **FilteredBookEventsOut** generates a tuple using the streams **xml** format allowing this to be parsed in later streams operators or saved as a file in the FileSink operator (**EventOut**)
- **FilteredJMSBookEvents** generates a tuple with a single rstring attribute which is used as the payload for a JMSSink operator (**JMSOut**). This JMS event can then be routed to an external system for correlation or for further action like making an offer in order to close the order.

An optional error port is also included that provides a stream of runtime errors which in this case is sent to a FileSink operator (**Error**) for later analysis.

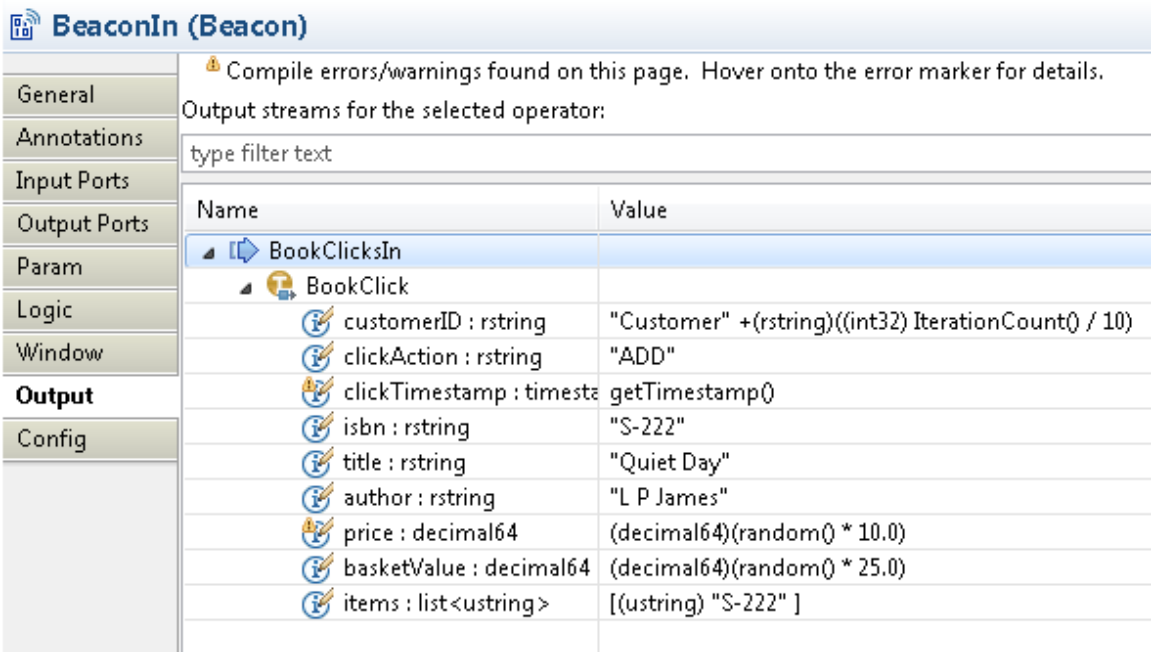
This pattern introduces the use of a Java XOM in ODM with a Book Java class and the use of customized mappings from ruleset parameters to output tuples routing to different output ports.

Each of the operators in the application is now described in more detail together with the mapping used in Ruleset Executor to produce the streams needed.

Beacon operator (BeaconIn)

BeaconIn is a Beacon operator that is used in this example to generate a stream of tuples with the same schema as the BookClickIn stream but with random price and basketValue values. The Beacon operator can be configured to generate values using the Output configuration shown in figure 4.2 below.

Figure 4.2 Beacon configuration to generate stream data



This stream uses the BookClick type to define the stream attributes. This is the same BookClickIn stream attributes for the Classification Pattern as tabled in 3.1. The values for price and basketValue are generated randomly to allow each tuple to have different classified results resulting in different events being generated according to the rules.

Warning messages (CDISP0079W) appear about multiple calls to a stateful function within the same output clause for getTimestamp and random functions. For a random generator, potential side effects are not an issue here in our sample run.

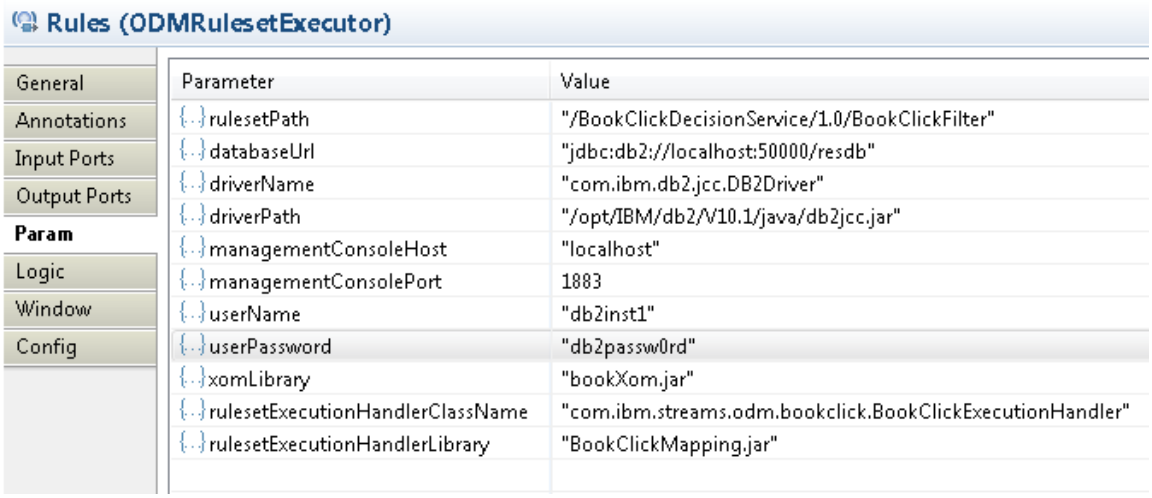
In this example, we set the iteration count to 100 and interval to 0.1 in the Beacon parameters so that 100 tuples are generated at 0.1 seconds interval. This allows us to check the expected number of events generated based on the number of tuples with

OFFERBASKET classification written to filteredBookClicksOut.txt. It will also allow the stream processing to run for at least 10 seconds which allows the ruleset execution to be observed in a Rule Execution server console.

ODMRulesetExecutor (Rules)

The BeaconIn operator stream is connected to the input port of the Rules operator. This is configured to load the BookClickFilter ruleset from the Decision Server repository and also to respond automatically to updates as described in the installation and configuration section earlier. The parameters settings for the ODMRulesetExecutor are summarized in figure 4.3 below.

Figure 4.3 Rules (ODMRulesetExecutor) configuration parameters



This configuration uses the latest deployed version of the BookClickFilter ruleset with interfaces defined by the decision service version BookClickDecisionService/1.0.

The parameters for connection to the Rule Execution Server console (management console) are the same as for the Classification pattern while the Rule Execution Server database is based on DB2 in this pattern to provide another topology configuration example.

In this pattern, a Ruleset execution handler, BookClickExecutionHandler, is used to perform customized mapping of the ruleset ouput parameters to the stream output tuple. This is configured with the ruleset execution handler class name and library in the parameters.

The BookClickFilter ruleset also uses a Book Java class as Java XOM. There are 2 options for configuring a Java XOM in the ODMRulesetExecutor. In Figure 4.3 above, the xomLibrary parameter configures the operator to load the bookXom.jar into the classpath. This library contains the Java XOM and needs to match the classes used in the ruleset for execution. The parameter can either be a full path or a relative path with

respect to the data directory of the project. In this example, we placed the bookXom.jar in the /FilteringPattern/data directory of the project.

Alternatively, if the Java XOM is a managed XOM deployed on the ODM RES database, then you can set the parameter managedXomDeployedOnDb to true instead of configuring the xomLibrary parameter. In either configuration, you still need to have a copy of the Java XOM in the streams studio environment to create the Java XOM object for the input parameters used for ODM ruleset execution.

After the rules have been processed, the registered BookClickExecutionHandler maps the output parameters returned from ODM to tuples on the different output ports for action.

Ruleset Executor Handler mapping to ruleset parameters

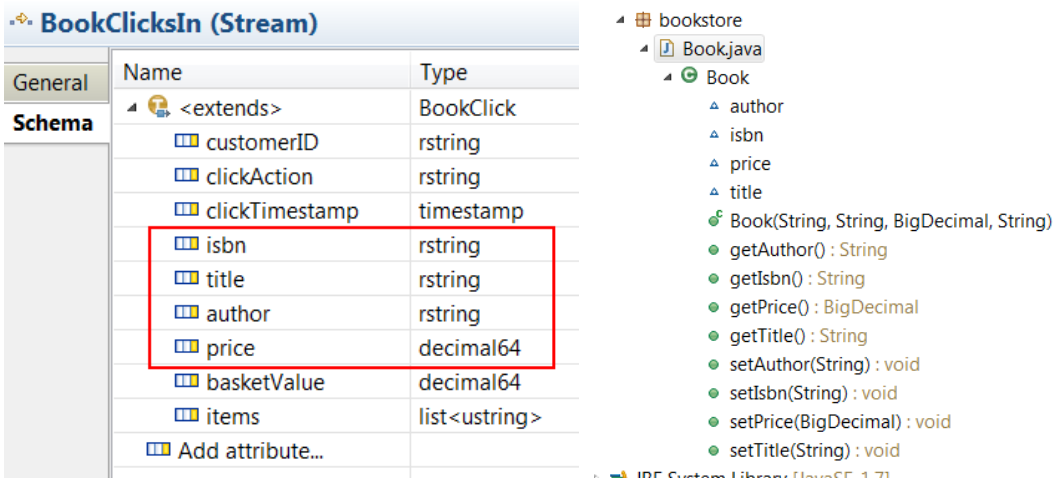
The ODMRulesetExecutor operator allows customized tuple mapping by using a ruleset executor handler. The com.ibm.streams.rules toolkit sample FeatureDemoCustomMapping shipped in the InfoSphere Stream is a good reference in this topic. In this pattern, BookClickExecutionHandler and BookClickMapping Java classes provide the custom mappings to forward different data to the different output ports.

The BookClickExecutionHandler class is invoked once for each tuple passed through the ruleset executor and provides the code to manage the overall mapping of ruleset parameters to operator ports. The BookClickMapping class provides the detailed mapping functions between the ruleset parameters and the stream on any given port.

The Java listings in this section should be studied with the ruleset execution handler in the FeatureDemoCustomMapping sample for a full picture.

The stream coming into the Ruleset Executor operator is the same as that provided for the classification pattern but the book fields (highlighted in red) need to be mapped into the Book Java class that will be sent to ODM as a parameter as shown in Figure 4.4.

Figure 4.4 Input stream and Book Java class



In this example, the BookClickFilter ruleset in BookClickDecisionService expects a Book Java XOM as one of the input parameters. The Book object is instantiated in the mapToInputParameters method in BookClickMapping class based on the data in the input tuple. The other input parameters, which are primitive attribute types, are mapped calling the super class method mapToInputParameters. This can be seen in listing 4.1 below.

Listing 4.1 BookClickExecutionHandler and BookClickMapping registration Java listing

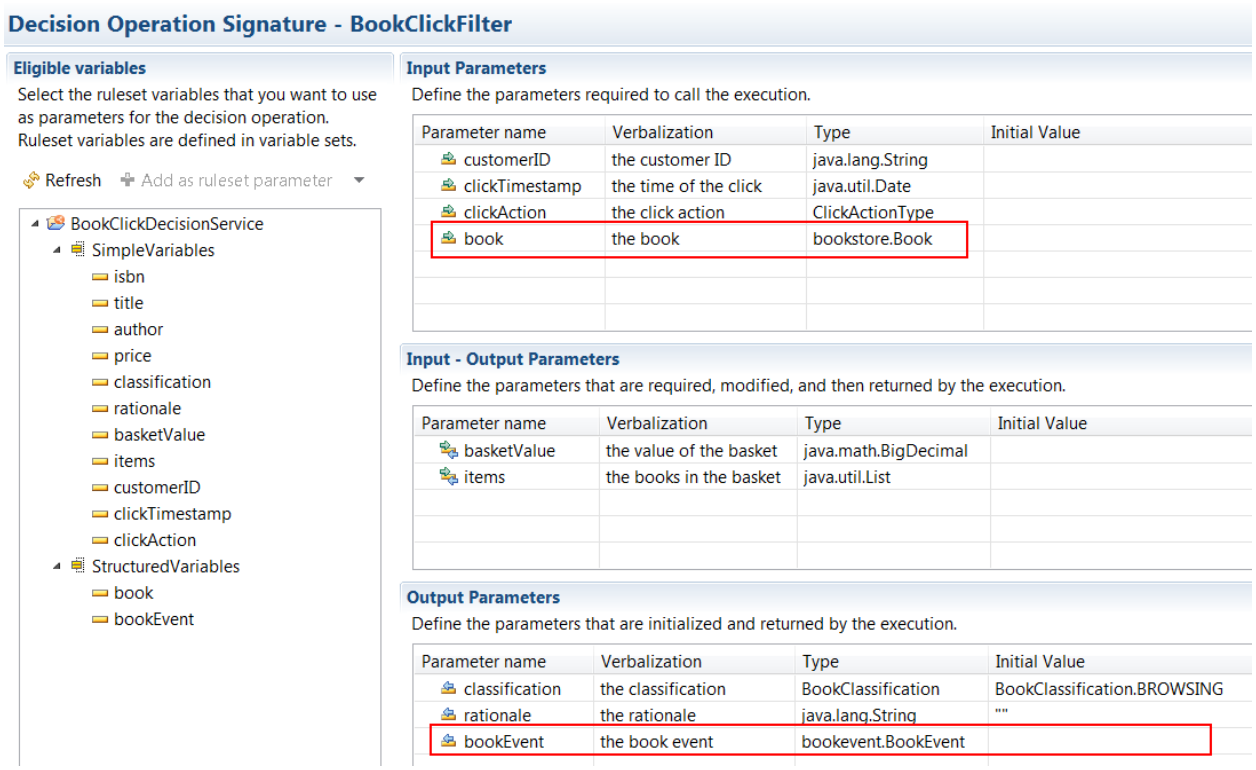
```
public Map<String, ?> mapToInputParameters(Tuple tuple) throws Exception {
	// Call to super will help user levarage the auto mapping for
	// primitive types
	Map<String, Object> inputParameters = (Map<String, Object>)
		super.mapToInputParameters(tuple);

	// Create the Book object of the Custom Defined types which need to
	// go as ODM Input Parameters from Beacon data
	String isbn = tuple.getString("isbn");
	String title = tuple.getString("title");
	BigDecimal price = tuple.getBigDecimal("price");
	String author = tuple.getString("author");

	Book book = new Book(isbn, title, price, author);
	inputParameters.put("book", book);

	return inputParameters;
}
```

Once the ruleset parameters have been created the RulesetExecutor operator then invokes the BookClickFilter ruleset on the JSE RES.

BookClickFilter Ruleset

The BookClickFilter decision operation signature can be defined based on a combination of the simple variables used by the BookClickClassification ruleset and the StructuredVariables based on the Book and BookEvent BOM. Figure 4.15 shows the BookClickFilter decision operation signature which can be compared with the BookClickClassification decision operation signature from Figure 3.2. A book parameter replaces the isbn, title, author and price parameters and a bookEvent parameter is added to the output parameters.

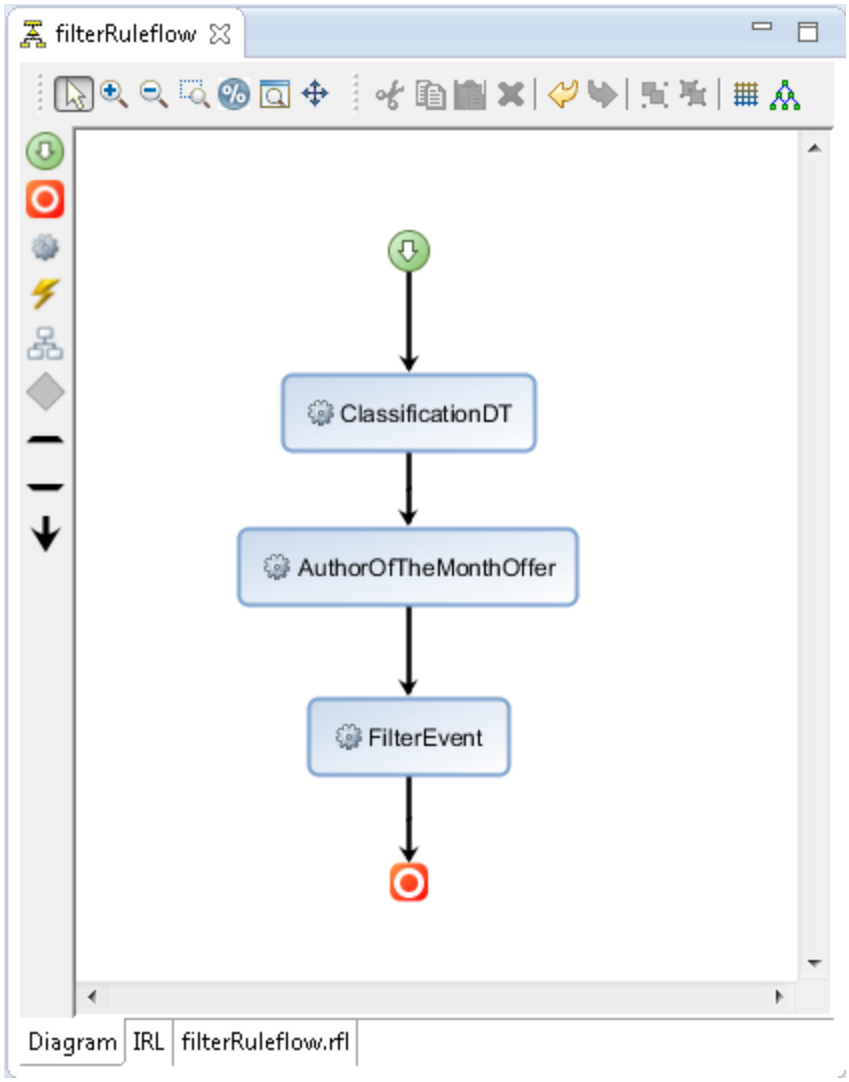
Figure 4.5 BookClickFilter Decision Operation signature



This signature shows the new book input parameter populated in the previous section together with the bookEvent output parameter that will be created if required by the rules.

With the BOM, decision service variables and parameters defined, the ruleset follows the Filter ruleflow as shown in figure 4.6 below.

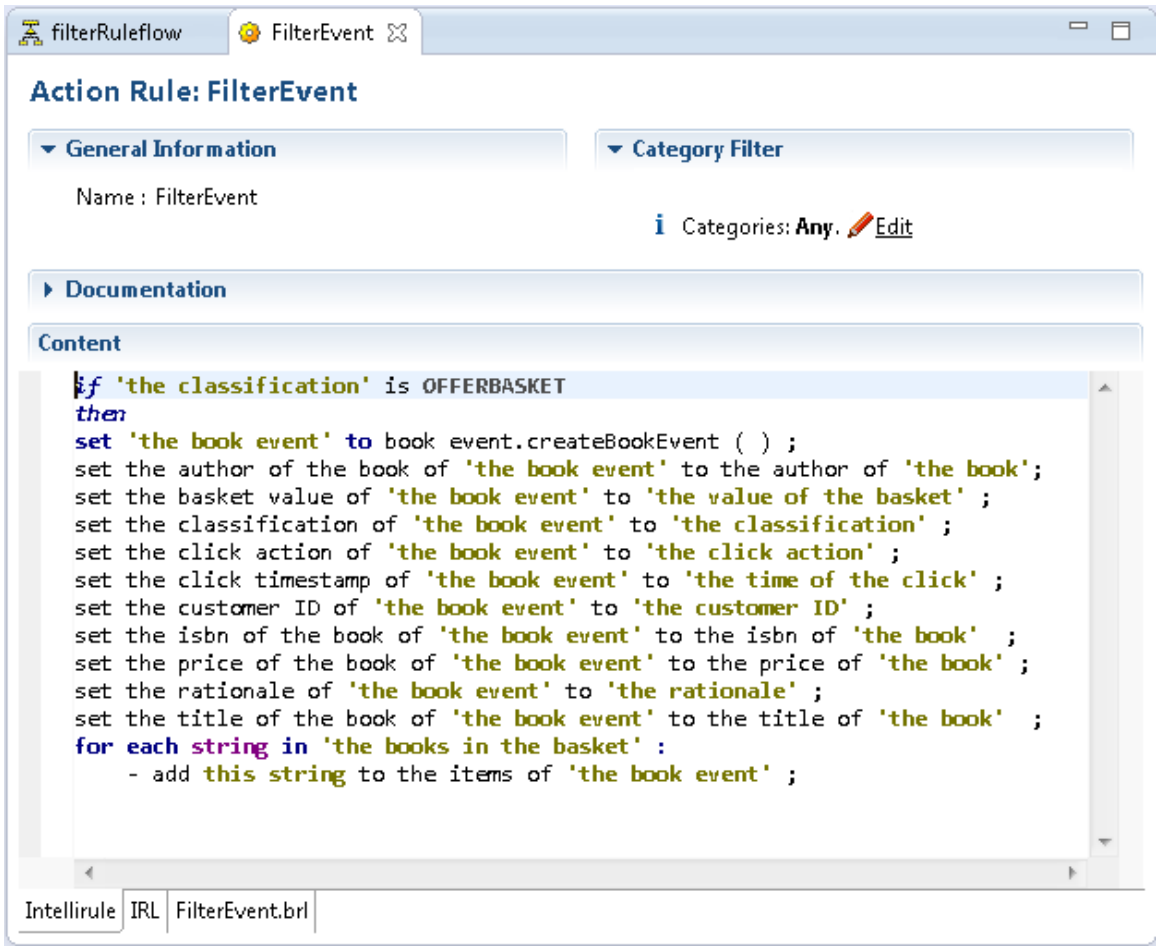
Figure 4.6 Filter Ruleflow



The ruleflow does not need to initialize the book variable as this has been provided as an input parameter. The flow applies the ClassificationDT decision table (figure 3.7) to classify the click and then the AuthorOf TheMonth (figure 3.8) rules to identify when an Offer should be made. These rules are the same as those used in the Classification ruleset.

The ruleflow then applies the FilterEvent rule which selects those clicks (tuples) that are classified as OFFERBASKET and creates a book event as shown in figure 4.7.

Figure 4.7 FilterEvent Action Rule



The book event is only created if the classification is OFFERBASKET. When this condition is met a BookEvent is created using the **createBookEvent()** operation that was defined in the BOM and the fields of the BookEvent initialized from the available variables. On completion of this rule the bookEvent variable is returned as an output parameter back to the RulesetExecutor stream operator.

This ruleset shows how to leverage the existing rules for classification and add the ability to identify significant events which can be returned as output parameters to the streams for further action. The pattern also shows how we introduce structure to the object model and map that to the streams data by using a Java XOM and or a schema based XOM. The next section shows how the BookClickFiltering ruleset return parameters are mapped back into the RulesetExecutor output streams.

Ruleset Executor Handler mapping to FileSink: CSVOut

CSVOut is a FileSink operator similar to the CSVOut operator in the Classification Pattern which writes the classified tuples output from the Rules operator port to a CSV file. When a ruleset execution handler is registered for the rule operator, a call is made

to the method **mapToTuple** for each set of returned parameters and each output port. The implementation in the case of the filtered output port (as the book object is not a return parameter) can use the default operation from the superclass to map all simple ruleset parameters to the stream tuples as shown in listing 4.2 below.

Listing 4.2 Default mapping from ruleset parameters to tuples

```
public void mapToTuple(StreamingOutput<OutputTuple> outputPort,
                    OutputTuple tuple, Map<String, ?> outMap) throws Exception
{
       // Mapping of Port 0 - main port filtered
       if (outputPort.getPortNumber() == 0) {
              // Call to super.mapToTuple(outputPort, tup, outMap);
              // is to help user leverage auto mapping of primitive types
              super.mapToTuple(outputPort, tuple, outMap);
       }
       ::::::
}
```

Ruleset Executor Handler mapping to FileSink: EventOut

EventOut is a FileSink operator used for writing the event data in XML format to a CSV file. The mapping takes the eventOut parameter returned from the ruleset (if present) and writes the xml string as a single xml attribute in the stream sent to the EventOut operator as shown in listing 4.3.

This demonstrates that events can be forwarded as XML to other operators in a streams application.

The bookEvent is an optional output parameter from the ruleset and is only sent when the tuple is classified as OFFERBASKET. This means that a check for the existence of a bookEvent needs to be undertaken in the handleExecutionResponse method in BookClickExecutionHandler before we call the data mapping for the event output ports. If this check is not performed, null or empty String data will be sent to the output port even when no event is generated. See listing 4.3 below on checking for bookEvent data and submitting events to event ports.

Listing 4.3 Checking for event parameter existence before mapping

```
String bookEventContent = (String) outMap.get("bookEvent");
       if (bookEventContent != null) {
              // Book event available, send to port 1 which is book event port
              StreamingOutput<OutputTuple> outputPort1 = getOperatorContext()
                                  .getStreamingOutputs().get(1);

              // Create Output Tuple for output port 1
              OutputTuple streamsOutputTuple1 = outputPort1.newTuple();
              streamsOutputTuple1.assign(inputTuple);

              try {
                     getTupleRulesetParamMapping().mapToTuple(outputPort1,
                                         streamsOutputTuple1, outMap);
```

```
        } catch (Exception e) {
              handleExecutionException(e, inputTuple, inputParameters);
              return;
        }

        // Submit book event content
        outputPort1.submit(streamsOutputTuple1);

        // Continue for port 2
        ::::::::::::
```

This switch is applied to both port 1 and port 2.

In the mapping class, the bookEvent output parameter from the rule execution is retrieved from a Map in String format which then needs to be converted to SPL XML format for the EventOut port. After conversion, we set the xmlEvent in the OutputTuple using setXML method. See Listing 4.4 below on the mapping done for port EventOut.

Listing 4.4 Mapping bookEvent from ruleset parameter to XML

```
public void mapToTuple(StreamingOutput<OutputTuple> outputPort,
                  OutputTuple tuple, Map<String, ?> outMap) throws Exception
{
       // Mapping of Port 0 - main port filtered
       ::::::

       // Mapping of Port 1 - book event returned as xml
       // to be logged in output file
       if (outputPort.getPortNumber() == 1) {
             String bookEventString = (String) outMap.get("bookEvent");
             XML xml = ValueFactory.newXML(
                   new ByteArrayInputStream( bookEventString.getBytes()));
             tuple.setXML("xmlEvent", xml);
       }
       ::::::
}
```

The Messaging Toolkit operators and the JMSSink operator in particular, requires a string to be sent as the message payload rather than an SPL XML type. In this case we need to map the bookEvent data into a String when sending the stream to JMSOut port. See listing 4.5 below on the mapping done for port JMSOut.

Listing 4.5 Mapping bookEvent from ruleset parameter to String

```
public void mapToTuple(StreamingOutput<OutputTuple> outputPort,
                  OutputTuple tuple, Map<String, ?> outMap) throws Exception
{
       // Mapping of Port 0 - main port filtered
       ::::::

       // Mapping of Port 1 - book event returned as xml
```

```
        // to be logged in output file
        ::::::

        // Mapping of Port 2 - book event returned as xml in String
        // to be forwarded as JMS message.
        if (outputPort.getPortNumber() == 2) {
              String bookEventString = (String) outMap.get("bookEvent");
              // Use of setString handles the conversion between
              // Java String and SPL rstring
              tuple.setString("bookEvent",  bookEventString);
        }
}
```

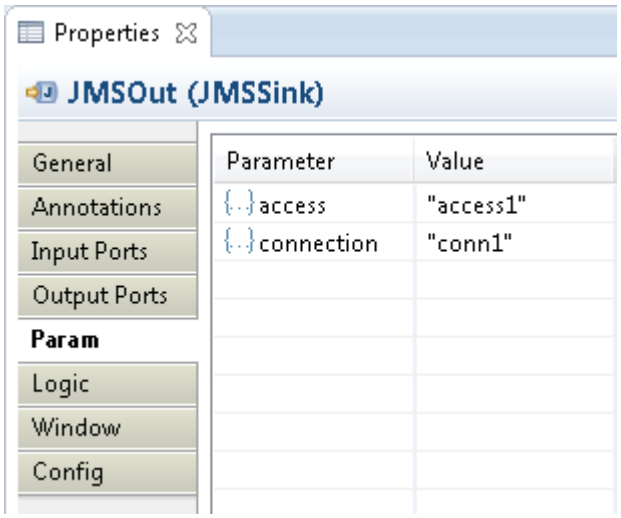
The JMSOut operator is expecting the bookEvent data as SPL rstring. The bookEvent data is extracted from the output parameter Map as a java.lang.String. The conversion from Java String to SPL rstring is handled by the setString method of the OutputTuple. If OutputTuple.setObject method is used instead, the Java String needs to be wrapped in a Java RString object.

JMSOut : JMSSink

JMSOut is a JMSSink operator from the Messaging Toolkit and is included in this application to show that the events identified by rules can be sent via messaging to remote systems for further action. The book event data is sent as a String whose content is an XML document that can be used as a payload. The Messaging Toolkit includes operators for sending messages using JMS, XMS and MQTT protocols. It is important to study the documentation in IBM Knowledge Centre for the Messaging Toolkit which has details on the use of the 3 different operator types, the data format supported and the SPL to JMS/XMS conversions.

In our example, we send the event as a JMS message using JMSSink to a WebSphere MQ queue. In the JMSSink operator, we need to set the parameters for the JMS connection and access for our JMS environment as shown in figure 4.8

Figure 4.8 JMSSink parameters



The access and connection values referred to are defined in a connection document. The default document location is Resources/etc/connections.xml in the project. If the connection document is in a different file location, then the full path to the connection document can be defined in the connectionDocument parameter for the JMSSink operator.

The connection configuration used in our sample environment (as described in the annex) is described in listing 4.5 below.

Listing 4.5 JMS connection configuration in connections.xml

```
<st:connections xmlns:st="http://www.ibm.com/xmlns/prod/streams/adapters"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <connection_specifications>
       <connection_specification name="conn1">
         <JMS initial_context="com.sun.jndi.fscontext.RefFSContextFactory"
              provider_url=file:///opt/JNDI-Directory
              connection_factory="ConnectionFactory" />
       </connection_specification>
  </connection_specifications>

  <access_specifications>
       <access_specification name="access1">
         <destination identifier="BOOKEVENT_IN"
              delivery_mode="persistent"
              message_class="xml" />
         <uses_connection connection="conn1"/>
         <native_schema>
              <attribute name="bookEvent" type="String" />
         </native_schema>
       </access_specification>
</access_specifications>

</st:connections>
```

The connection_specification_name (conn1) and the access_ specification_name (access1) defined in the connections.xml are used in setting the parameters in the JMSOut port. The JMS to SPL type mapping is based on the message_class defined in the access destination. See Attribute element for the list of message_classes supported and the mappings between message_class and the attribute type. The link also describes the optional length attribute for each operator. Depending on the operator type, message_class and attribute type, you may need to add the length attribute to avoid truncation of data.

In our sample, we set the message_class to xml. Note that xml SPL type is not supported by the Messaging Toolkit and we set the attribute type for bookEvent to **String** in Listing 4.5 above. If you intend to send the event to the ODM event runtime (previously called WebSphere Business Events (WBE)), you can set the message_class to "wbe". The JMS message will then have WBE-related headers included.

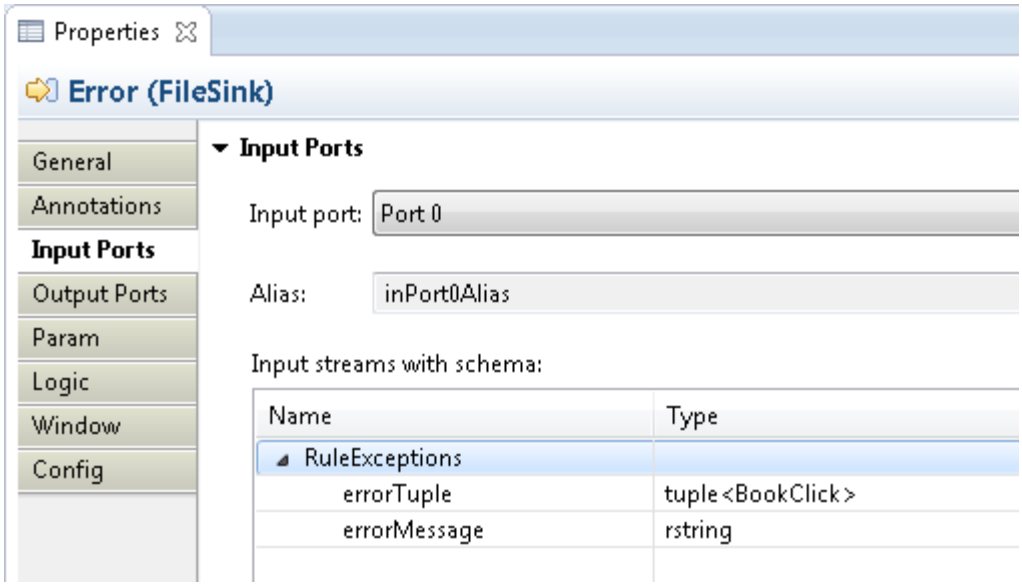
The Redbook on IBM InfoSphere Streams: Accelerating Deployments with Analytic Accelerators has details on configurations for the Messaging Toolkit. It also provides shortcuts for bypassing WMQ related permissions to simplify the running of samples.

For more details of configuring streams to support JMS messaging refer to Annex B.

Ruleset Execution Error handling through FileSink: Error

The RulesetExecution operator provides an additional port that can be used to provide a stream of exceptions that occur when processing a tuple through ODM. This Error port outputs a stream that is connected to the FileSink (Error) operator which is configured to record any errors to file Errors.txt. The error stream schema is shown below in figure 4.9.

Figure 4.9 Error port schema



The streams schema includes errorTuple and the errorMessage. The order of the schemas is important. Swapping the 2 error types will cause the Exceptions as shown in Listing 4.8 below and the streams fail to run.

Listing 4.8 Optional error output port attributes ordering

```
16 Oct 2014 17:17:05.310 [12112] ERROR #splapplog,J[0],P[0],Rules
M[JavaOp.cpp:log:92]  - CDIST2257E The second attribute in the optional error
output port must be an rstring.
16 Oct 2014 17:17:05.310 [12112] ERROR #splapplog,J[0],P[0],Rules
M[JavaOp.cpp:log:92]  - CDIST3358E The first attribute in the optional error
output port must be a tuple.
```

There should not be any records in Errors.txt for successful runs.

The default port for error output is port 1 with port 0 being the main application port. The Filtering pattern has a total of 4 ports including the Error port. In this example, we set the error port to the last as in the FeatureDemoCustomMapping sample.

FilteringPattern Listing

The complete listing for the FilteringPattern application is provide in listing 4.9 below.

Listing 4.9 FilteringPattern application SPL listing

```
namespace application ;

use com.ibm.streams.rules.odm::ODMRulesetExecutor ;
use com.ibm.streams.messaging.jms::JMSSink ;
```

```
composite FilteringPattern
{
    type
        BookClick = rstring customerID, rstring clickAction,
            timestamp clickTimestamp, rstring isbn, rstring title, rstring
author,
            decimal64 price, decimal64 basketValue, list<ustring> items ;
        FilteredBookClick = rstring customerID, rstring clickAction,
            timestamp clickTimestamp, rstring isbn, rstring title, rstring
author,
            decimal64 price, decimal64 basketValue, list<ustring> items,
            rstring classification, rstring rationale ;
    graph
        (stream<BookClick> BookClicksIn) as BeaconIn = Beacon()
        {
            param
                iterations : 100 ;
                period : 0.1 ;
            output
                BookClicksIn : customerID = "Customer" +(rstring)((int32)
IterationCount()
                    / 10), clickTimestamp = getTimestamp(), isbn = "S-222",
title =
                    "Quiet Day", author = "L P James", price =
(decimal64)(random() * 10.0),
                    basketValue = (decimal64)(random() * 25.0), items =
[(ustring) "S-222" ],
                    clickAction = "ADD" ;
        }

        (stream<FilteredBookClick> FilteredBookClicksOut as outPort0Alias ;
            stream<xml xmlEvent> FilteredBookEventsOut as outPort1Alias ;
            stream<rstring bookEvent> FilteredJMSBookEvents as outPort2Alias ;
            stream<tuple<BookClick> errorTuple, rstring errorMessage>
RuleExceptions as
            outPort3Alias) as Rules = ODMRulesetExecutor(BookClicksIn as
inPort0Alias)
        {
            param
                rulesetPath : "/BookClickDecisionService/1.0/BookClickFilter"
;
                databaseUrl : "jdbc:db2://localhost:50000/RESDB86" ;
                driverName : "com.ibm.db2.jcc.DB2Driver" ;
                driverPath : "/opt/IBM/db2/V10.1/java/db2jcc.jar" ;
                managementConsoleHost : "localhost" ;
                managementConsolePort : 1883 ;
                userName : "db2inst1" ;
                userPassword : "db2passw0rd" ;
                xomLibrary : "bookXom.jar" ;
                rulesetExecutionHandlerClassName :
                    "com.ibm.streams.odm.bookclick.BookClickExecutionHandler"
;
                rulesetExecutionHandlerLibrary : "BookClickMapping.jar" ;
        }
```

```
        () as CSVOut = FileSink(FilteredBookClicksOut as inPort0Alias)
        {
            param
                file : "filteredBookClicksOut.txt" ;
                format : csv ;
        }

        () as EventOut = FileSink(FilteredBookEventsOut as inPort0Alias)
        {
            param
                file : "filteredBookEventsOut.txt" ;
                format : csv ;
        }

        () as Error = FileSink(RuleExceptions as inPort0Alias)
        {
            param
                file : "Errors.txt" ;
                format : csv ;
        }

        () as JMSOut = JMSSink(FilteredJMSBookEvents as inPort0Alias)
        {
            param
                access : "access1" ;
                connection : "conn1" ;
        }

}
```

Filtering Pattern Execution

The filtering pattern sample may be executed in a standalone stream to demonstrate the integration techniques. The multiple output ports have different output data in different formats.

The output from CSVOut port is very similar to the CSVOut in classification pattern. With the Beacon generating 100 random tuples, the CSVOut should have 100 records with random data as shown in listing 4.11.

Listing 4.10 filteredBookClicksOut.txt Sample output stream in csv format

```
"Customer0","ADD",(1413458660,134793000,0),"S-222","Quiet Day","L P
James",0.1773576019331813,8.868423302192241,["S-222"],"FILLINGBASKET","Items in basket."
"Customer0","ADD",(1413458660,260986000,0),"S-222","Quiet Day","L P
James",0.217961915768683,21.45777679979801,["S-222"],"OFFERBASKET","High value basket: Offer AOTM to close
order."
"Customer0","ADD",(1413458660,959895000,0),"S-222","Quiet Day","L P
James",6.52473428286612,15.69889669772238,["S-222"],"FILLINGBASKET","Items in basket."
```

```
"Customer0","ADD",(1413458661,65070000,0),"S-222","Quiet Day","L P
James",3.213561940938234,7.297414459753782,["S-222"],"FILLINGBASKET","Items in basket."
"Customer0","ADD",(1413458661,176415000,0),"S-222","Quiet Day","L P
James",8.917076834477484,6.223083520308137,["S-222"],"FILLINGBASKET","Items in basket."
"Customer0","ADD",(1413458661,281345000,0),"S-222","Quiet Day","L P
James",7.776940008625388,22.92293792124838,["S-222"],"OFFERBASKET","High value basket: Offer AOTM to close
order."
"Customer0","ADD",(1413458661,393003000,0),"S-222","Quiet Day","L P
James",7.02701048925519,18.24273504316807,["S-222"],"FILLINGBASKET","Items in basket."
"Customer0","ADD",(1413458661,497381000,0),"S-222","Quiet Day","L P
James",4.393221000209451,6.449947797227651,["S-222"],"FILLINGBASKET","Items in basket."
"Customer0","ADD",(1413458661,601683000,0),"S-222","Quiet Day","L P
James",8.860875847749412,3.423841251060367,["S-222"],"FILLINGBASKET","Items in basket."
"Customer0","ADD",(1413458661,705928000,0),"S-222","Quiet Day","L P
James",5.077155292965472,24.38116824487224,["S-222"],"OFFERBASKET","High value basket: Offer AOTM to close
order."
"Customer1","ADD",(1413458661,815945000,0),"S-222","Quiet Day","L P
James",9.768905667588115,5.983907252084464,["S-222"],"FILLINGBASKET","Items in basket."
::::::::::::::::::::
```

As the filter event rule sends book event based on the classification, the number of book events received in the other ports should be the number of records with OFFERBASKET classification in filteredBookClicksOut.txt which is random for each run with random input tuples. For this particular run, there are 19 tuples with OFFERBASKET classification.

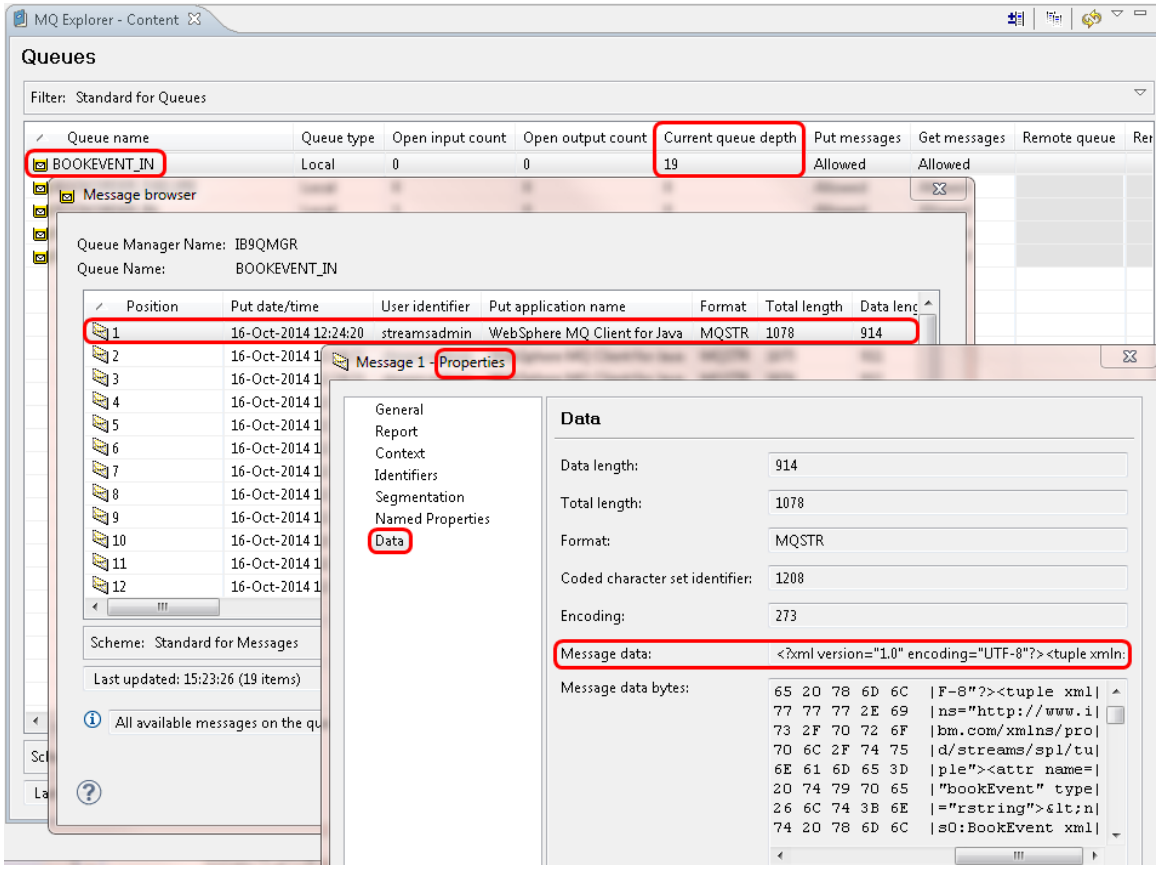
The book event output parameter is of XML kind and can sent as an XML to target destinations. In this example, the xml formatted book event tuples are written to a file filteredBookEventsOut.txt. For this run, there are 19 book events tuples. The Listing 4.11 below shows the 3 book events corresponding to the 3 records with OFFERBASKET classification shown in Listing 4.11 above.

Listing 4.11 filteredBookEventsOut.txt Sample output tuples in xml format

```
"<ns0:BookEvent xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:ns0=\"http://BookEvent\"><ns0:customerID>Customer0</ns0:customerID><ns0:clickTimestamp>2014-10-
16T13:24:20.260</ns0:clickTimestamp><ns0:clickAction>ADD</ns0:clickAction><ns0:basketValue>21.457777</ns0:basket
Value><ns0:classification>OFFERBASKET</ns0:classification><ns0:rationale>High value basket: Offer AOTM to close
order.</ns0:rationale><ns0:book><ns0:isbn>S-222</ns0:isbn><ns0:title>Quiet Day</ns0:title><ns0:author>L P
James</ns0:author><ns0:price>0.21796192</ns0:price></ns0:book><ns0:items>S-222</ns0:items></ns0:BookEvent>"x
"<ns0:BookEvent xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:ns0=\"http://BookEvent\"><ns0:customerID>Customer0</ns0:customerID><ns0:clickTimestamp>2014-10-
16T13:24:21.281</ns0:clickTimestamp><ns0:clickAction>ADD</ns0:clickAction><ns0:basketValue>22.922937</ns0:basket
Value><ns0:classification>OFFERBASKET</ns0:classification><ns0:rationale>High value basket: Offer AOTM to close
order.</ns0:rationale><ns0:book><ns0:isbn>S-222</ns0:isbn><ns0:title>Quiet Day</ns0:title><ns0:author>L P
James</ns0:author><ns0:price>7.77694</ns0:price></ns0:book><ns0:items>S-222</ns0:items></ns0:BookEvent>"x
"<ns0:BookEvent xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:ns0=\"http://BookEvent\"><ns0:customerID>Customer0</ns0:customerID><ns0:clickTimestamp>2014-10-
16T13:24:21.705</ns0:clickTimestamp><ns0:clickAction>ADD</ns0:clickAction><ns0:basketValue>24.381168</ns0:basket
Value><ns0:classification>OFFERBASKET</ns0:classification><ns0:rationale>High value basket: Offer AOTM to close
order.</ns0:rationale><ns0:book><ns0:isbn>S-222</ns0:isbn><ns0:title>Quiet Day</ns0:title><ns0:author>L P
James</ns0:author><ns0:price>5.077155</ns0:price></ns0:book><ns0:items>S-222</ns0:items></ns0:BookEvent>"x
::::::::::::::::
```

The JMSOut port should be sending the same number of book events to the configured JMS destination BOOKEVENT_IN as JMS messages. With the MQ Explorer, we can browse the JMS message arrived in the BOOKEVENT_IN queue and view the data sent as shown in Figure 4.16 below.

Figure 4.10 JMS message browsing in MQ Explorer



If you find that the message is truncated in the MQ Explorer, you would need to define the length attribute for the native_schema in the connections.xml discussed earlier. Just make sure that the MQ Explorer Preferences under **WebSphere MQ Explorer > Messages > Max data bytes displayed** is not causing the truncation in the UI first.

The Listing 4.13 below shows the message data for the first JMS message when message_class is set to "xml"

Listing 4.12 JMS Message for xml message_class

```
<?xml version="1.0" encoding="UTF-8"?><tuple xmlns="http://www.ibm.com/xmlns/prod/streams/spl/tuple"><attr
name="bookEvent" type="rstring">&lt;ns0:BookEvent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://BookEvent"&gt;&lt;ns0:customerID&gt;Customer0&lt;/ns0:customerID&gt;&lt;ns0:clickTimestamp&gt;
2014-10-
16T13:24:20.260&lt;/ns0:clickTimestamp&gt;&lt;ns0:clickAction&gt;ADD&lt;/ns0:clickAction&gt;&lt;ns0:basketValue&
gt;21.457777&lt;/ns0:basketValue&gt;&lt;ns0:classification&gt;OFFERBASKET&lt;/ns0:classification&gt;&lt;ns0:rati
onale&gt;High value basket: Offer AOTM to close order.&lt;/ns0:rationale&gt;&lt;ns0:book&gt;&lt;ns0:isbn&gt;S-
222&lt;/ns0:isbn&gt;&lt;ns0:title&gt;Quiet Day&lt;/ns0:title&gt;&lt;ns0:author&gt;L P
James&lt;/ns0:author&gt;&lt;ns0:price&gt;0.21796192&lt;/ns0:price&gt;&lt;/ns0:book&gt;&lt;ns0:items&gt;S-
222&lt;/ns0:items&gt;&lt;/ns0:BookEvent&gt;</attr></tuple>
```

For comparison, Listing 4.44 shows a JMS message for a bookEvent data using wbe as message_class from a different run.

Listing 4.13 JMS Message for wbe message_class

```
<?xml version="1.0" encoding="UTF-8"?><connector name="System S" version="6.2"
xmlns="http://wbe.ibm.com/6.2/Event/inPort0Alias"><connector-bundle name="inPort0Alias"
type="Event"><inPort0Alias><bookEvent data-type="string">&lt;ns0:BookEvent
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns0="http://BookEvent"&gt;&lt;ns0:customerID&gt;Customer0&lt;/ns0:customerID&gt;&lt;ns0:clickTimestamp&gt;
2014-10-
16T17:49:57.552&lt;/ns0:clickTimestamp&gt;&lt;ns0:clickAction&gt;ADD&lt;/ns0:clickAction&gt;&lt;ns0:basketValue&
gt;22.52891&lt;/ns0:basketValue&gt;&lt;ns0:classification&gt;OFFERBASKET&lt;/ns0:classification&gt;&lt;ns0:ratio
nale&gt;High value basket: Offer AOTM to close order.&lt;/ns0:rationale&gt;&lt;ns0:book&gt;&lt;ns0:isbn&gt;S-
222&lt;/ns0:isbn&gt;&lt;ns0:title&gt;Quiet Day&lt;/ns0:title&gt;&lt;ns0:author&gt;L P
James&lt;/ns0:author&gt;&lt;ns0:price&gt;2.2024555&lt;/ns0:price&gt;&lt;/ns0:book&gt;&lt;ns0:items&gt;S-
222&lt;/ns0:items&gt;&lt;/ns0:BookEvent&gt;</bookEvent></inPort0Alias></connector-bundle></connector>
```

The JMS messages in the BOOKEVENT_IN queue can now be consumed by applications, e.g. IIB, WBE, waiting on the queue to act on the book events.

Summary

We have shown in this section how to build on the simpler Classification Pattern to create a new pattern that introduces improved data structure and creating events that can be sent to different target destinations like JMS. The pattern integrates InfoSphere Streams and ODM using the ODMRulesetExecutor from the Rules Toolkit to execute a deployed BookClickFilter ruleset to classify a book click action and create an event for classifications that requires further actions. Further more, it is possible to customize the data mapping using the ruleset execution to send data to different ports expecting different data formats using the ruleset execution handler supported by the Rules Toolkit. The same data can be sent to more than one destination for different analysis, actions as required.

Conclusion

This article has examined how Operational Decision Management techniques can be used within Big Data solutions such as streams. The goal of this integration is to leverage rule and policy based categorization of the information to assist in the analytics and identify information of significance to emerging business situations. This allows business decisions to respond dynamically to the evolving situations providing an optimized response in the complex emerging business environment.

Annex A: Installation and Configuration of InfoSphere Streams 3.2.1 with IBM Operational Decision Manager

This section describes how to install and configure InfoSphere streams with IBM Operational Decision Manager to support the use of business rules with streams

applications. The tutorial is based on the use of the InfoSphere Streams Quick Start Edition which is available for non-production environments as a VMWare Image or as a native Linux install. The version used for this tutorial is v3.2.1.
This section also includes instructions on running the ODM samples provided with IBM InfoSphere streams.

InfoSphere Streams installation and configuration overview

This section provides an overview of how to establish an InfoSphere streams environment. Each subsection references the recommended InfoSphere documentation to be followed to perform that task.

InfoSphere Streams Installation

http://www-01.ibm.com/support/knowledgecenter/SSCRJU_3.2.1/com.ibm.swg.im.infosphere.streams.install.doc/doc/install-container.html?lang=en

This tutorial recommends the use of the Quick start edition as this provides a readily available install for describing the integration patterns with ODM. The ODM integration patterns are then also applicable to more sophisticated InfoSphere streams installations where administrators understand the details of the topologies to be established.

The VMWare image provides a preconfigured installation that is ideal for prototypes. The native Linux install can be installed through a GUI (if an X Windows System is installed on your host) or through an interactive console mode. While you can install and use the Quick Start Edition without a GUI (for example as a server), the GUI allows you to get started more quickly and use the Streams Studio application development interface.

InfoSphere Streams Configuration

http://www-01.ibm.com/support/knowledgecenter/SSCRJU_3.2.1/com.ibm.swg.im.infosphere.streams.cfg.doc/doc/ibminfospherestreams-containercfg.html?lang=en

If you installed the VMware image, this configuration has been already undertaken and you can move onto configuring ODM in the environment or setting up Streams Studio on a remote workstation.

On a native install you should go through the post install configuration steps described here. http://www-01.ibm.com/support/knowledgecenter/api/content/SSCRJU_3.2.1/com.ibm.swg.im.infosphere.streams.install-admin.doc/doc/ibminfospherestreams-install-postinstall-roadmap.html
It is important that you follow these steps as they will be needed when you come to install and configure the ODM Operator.

For the native Linux install you need to undertake the configuration using the FirstSteps scripts. If you are using the GUI and **Launch First Steps** is selected in the

PostInstallation Tasks panel**,** the IBM InfoSphere Streams First Steps GUI starts at end of installation. The First Steps GUI can also be started later by running command *Streams-installation-directory*/FirstSteps.sh.

The First Steps documentation and GUI take you through the following important steps:

- Configure the SSH environment
  - Select **DSA** or **RSA SSH** key type.
    - In our environment, we select **RSA SSH** key type and proceeded to configure the optional **Generate public and private keys**.
- Configure InfoSphere Streams environment variables
  - Follow instructions provided to set the environment variables for the InfoStream user, i.e., add the following command to ~/.bashrc file or /etc/profile.d script:
    - source /opt/ibm/InfoSphereStreams/bin/streamsprofile.sh
- Verify the installation.
  - Although this is marked as optional. It is a good idea to verify the installation before proceeding to other tasks.
- Create and manage InfoSphere Streams instances
  - Select **Share the instance** if instance is to be used by other users.
  - Click **Check port availability** for the **SWS HTTPS port** (default 8443) before proceeding

InfoSphere Streams Studio Configuration

http://www-01.ibm.com/support/knowledgecenter/api/content/SSCRJU_3.2.1/com.ibm.swg.im.infosphere.streams.cfg.doc/doc/remote-development-creating-connection-linux.html

If you installed the VMware image, this configuration has been already undertaken and you can move onto installing and configuring ODM in the environment.

You can install InfoSphere Streams Studio locally or on a remote Windows or Linux workstation according to these instructions.

The installable images are shipped in *Streams-installation-directory*/etc/StreamsStudio:

- Windows: StreamsStudio-Win.zip
- Linux: StreamsStudio.zip

Copy and unzip the installable image on the platform of choice.

The InfoSphere Streams Studio requires a 64-bit Java Development Kit (JDK) with an IBM ORB. If you use an Oracle JDK, the Java installation must be configured to use the IBM ORB implementation. For more information, see Configuring an Oracle Java development kit for Streams Studio.

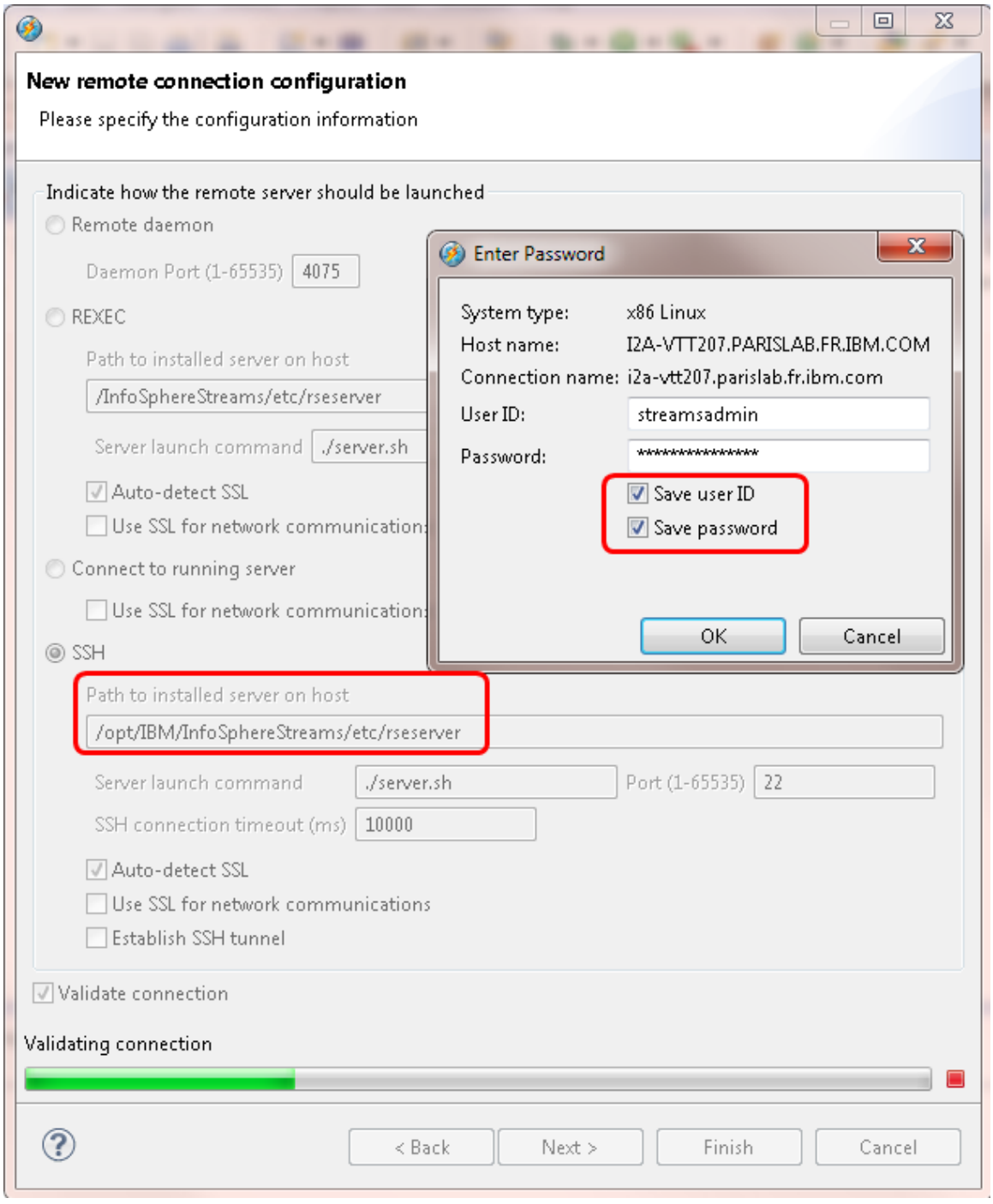
To re-configure the studio to use the correct JDK after studio is installed you must define where the JDK is installed in the file *StreamsStudio-installation-directory*/streamsStudio.ini.
Here is an example to configure a Windows installation of Steams Studio to use the IBM JDK that is shipped in the Operational Decision Manager (ODM) V8.6 64-bit installation:

-vm
C:\IBM\ODM86\jdk\bin\javaw.exe
-vmargs

Once the JAVA configuration is completed, you can now bring up the InfoSphere Streams Studio with the *StreamsStudio-installation-directory*/streamsStudio.exe command. On first run, you are prompted by the wizard to connect to the remote streams instance and set up both remote and local workspaces. These workspaces are synchronized and should use empty clean workspaces to start with. Existing projects can be imported into workspace after configuration.

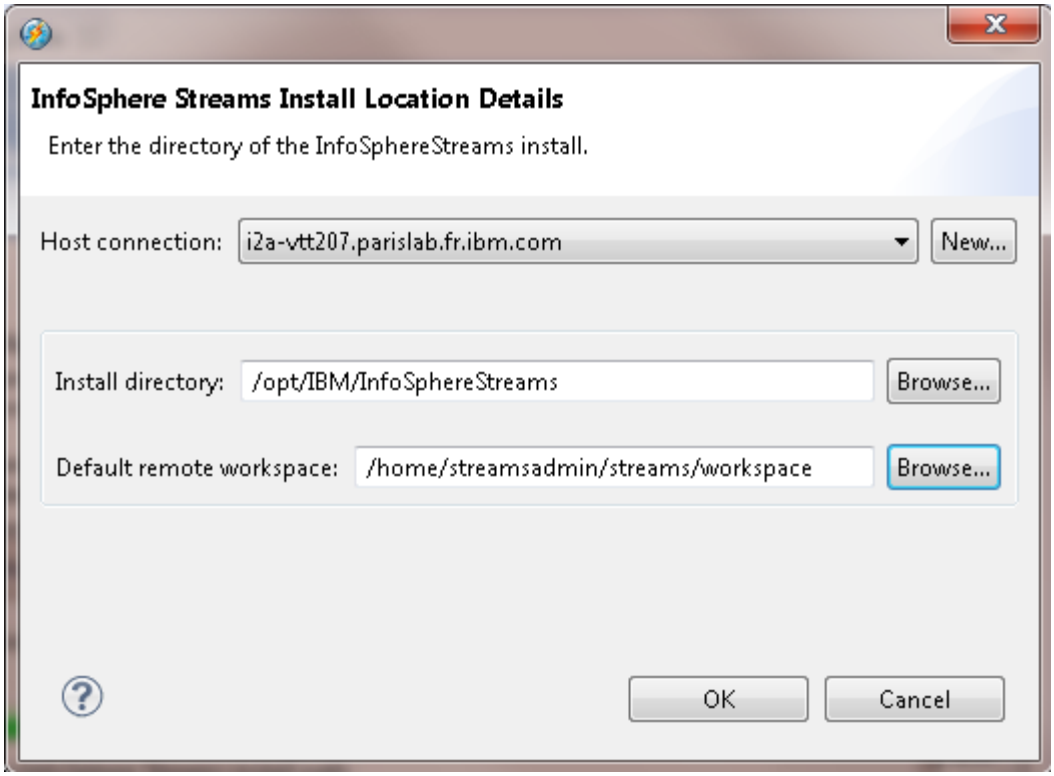
In the **New remote connection configuration**, the **SSH** option is selected in our environment with the **Path to installed server on host** modified with the correct installed directory. On clicking **Next**, enter the user id and password of a user on the remote host who can access the streams instance and select to save both user id and password.

Figure A.1 Remote connection configuration for Streams Studio



The wizard then prompts for the Infosphere Streams install directory and the default remote workspace for synchronizing with the local workspace. You can browse the remote file system with the Browse button.

Figure A.2 InfoSphere Streams install location details



This completes the InfoSphere Streams Studio configuration. Samples or existing projects can now be imported into the studio.

Configuring IBM Operational Decision Manager in a Streams Environment

http://www-01.ibm.com/support/knowledgecenter/api/content/SSCRJU_3.2.1/com.ibm.swg.im.infosphere.streams.rules-toolkit.doc/doc/container.html

Installation of ODM into a streams environment consists of the following steps.

1. Ensure the STREAM_INSTALL environment variable is setup (usually opt/ibm/InfoSphereStreams)
2. Install ODM Decision Server Rules. This does not have to be on the same host as streams but should be reachable over the network. This needs to have a network reachable database so that the embedded rule engine operating in streams can be populated dynamically with the rulesets. This tutorial describes how to setup this environment with a local tomcat Rule execution server console and a network derby database.
3. Copy the directory *ODM-installation-directory*/executionserver/lib to a directory reachable by streams. InfoSphere Streams expects the ODM J2SE jars to be

located in the ODM_HOME/executionserver/lib directory. You will need to ensure that the ODM_HOME variable is correctly configured to reflect this directory. (typically /opt/IBM/ODM*Version*, e.g. /opt/IBM/ODM86)

Installing Decision Server on Tomcat and Derby

This section provides a summary of the steps required to setup a lightweight local Decision Server running on Apache Tomcat. This is described in the Configuring Rule Execution Server on Tomcat section in the IBM Operational Decision Manager Knowledge Center.

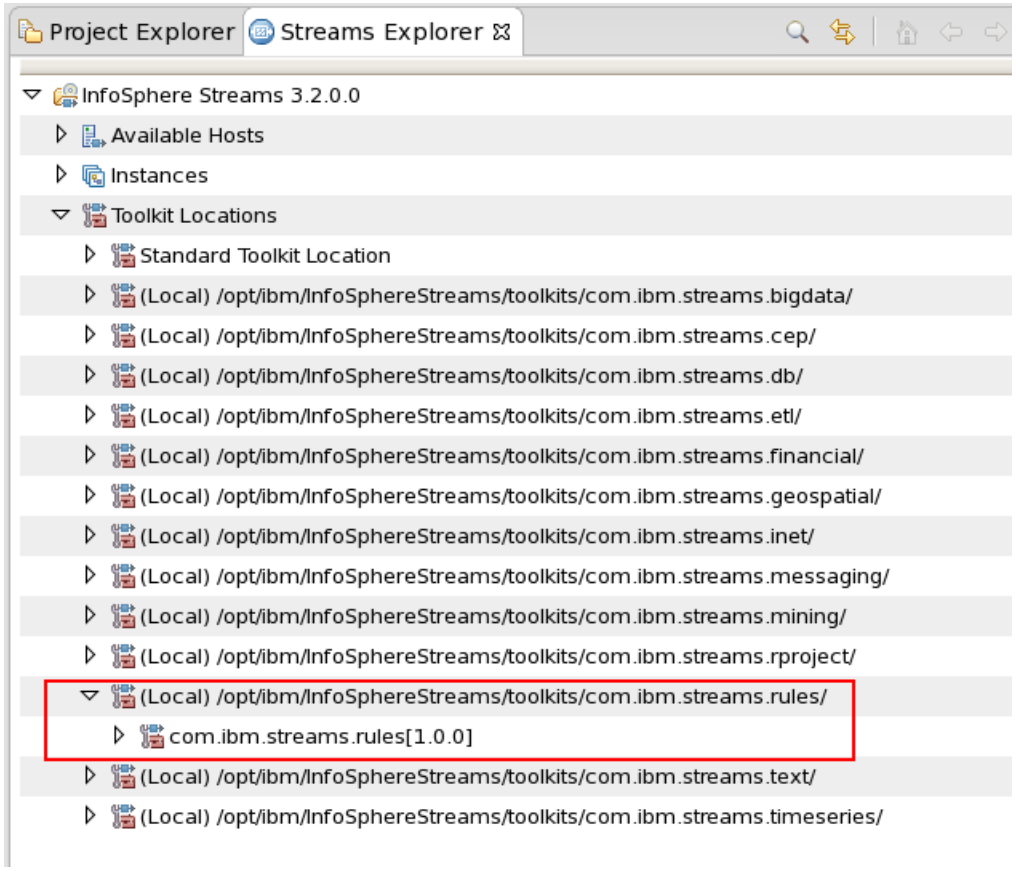
The following steps should be taken:

1. Install Apache Tomcat
2. Install Apache Derby a configure and start a network Derby server.
3. Add the derbyclient.jar library from the <DERBY_HOME>/lib directory to the <TOMCAT_HOME>/lib directory to ensure it is on the classpath. Note where this jar is located as it will need to be added to the configuration of your ODM
4. Install Decision Server (Rule Execution server) into Tomcat as described in the IBM Operational Decision Manager Knowledge Center here.
5. You should ensure that you use a network enabled database (e.g. DB2 or Network Derby) to allow the JSE RES used by the Streams ODM Operator to access it. The embedded derby database installed by default is not suitable.
6. You should also ensure that the Rule Execution Server console is enabled for TCPIP notification as described here in Changing the default behavior of the Management Console. The approach for repackaging the Tomcat management console war is different and is described here.

Configuring Streams Studio to use the Rules Toolkit

To allow the use of the Rules toolkit in streams studio you should add the toolkit location. This can be undertaken in the First Steps Task Launcher for Big Data in the Streams Studio by selecting **Make SPL Toolkits available**. This should then be visible in the streams explorer as shown below in figure A.3.

Figure A.3 Steams Explorer showing Rules Toolkit location



Building and running the Rules Toolkit sample applications

http://www-01.ibm.com/support/knowledgecenter/api/content/SSCRJU_3.2.1/com.ibm.swg.im.infosphere.streams.rules-toolkit.doc/doc/bldsamples.html

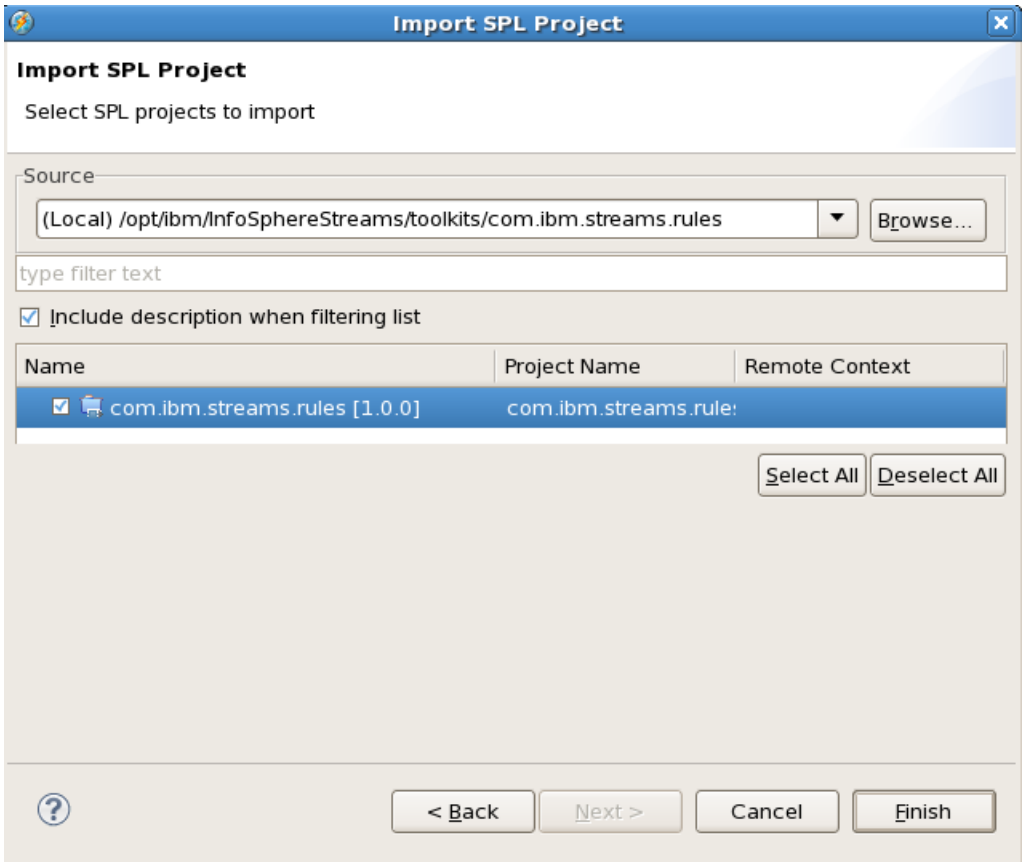
You can either import the whole Rules Toolkit into the workspace, or import individual samples.

To import the Rules Toolkit as an SPL project into the workspace:

1. Click **File > Import > InfoSphere Streams Studio > SPL Project.**
2. Click **Next**.
3. Click **Browse** to select a directory that contains the SPL toolkit. An SPL toolkit is identified by its model file, info.xml. Any directory containing this file is treated as the root of the SPL toolkit. If you are working in a remote development environment, you can import SPL projects either from your local system or from a remote Linux system. To import a project from a remote system, in the **Select a source location for import** window, from the **Connection list**, select the connection that you can use to connect to the remote system.

4. The **Import SPL Project** window lists all the SPL toolkits found from the selected directory. Each entry in this list shows the directory name, the toolkit name and the version. Select one or more toolkit(s) to import into the workspace. To select multiple toolkits, hold down the CTRL key while making your selection.
5. Click Finish.

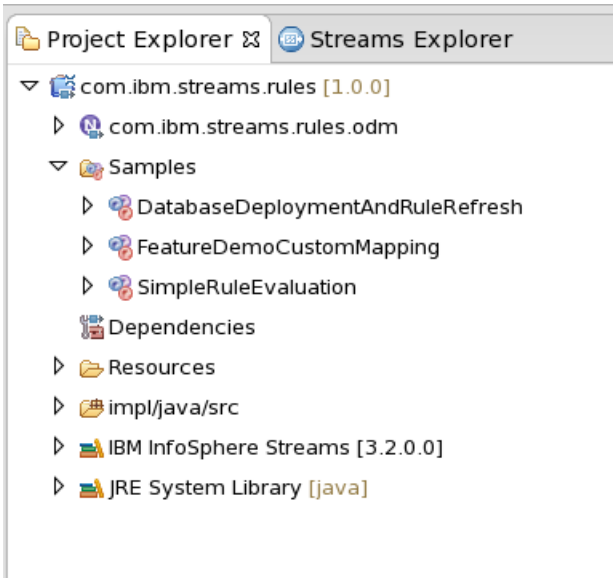
Figure A.4 Importing the toolkit into the workspace



When prompted on whether you want to delete unreferenced build configurations, click **Yes** to delete them to be rebuilt later.

On completion the project should be built and be visible in the project explorer as shown below in figure A.5.

Figure A.5 Project explorer after import of Rules toolkit.



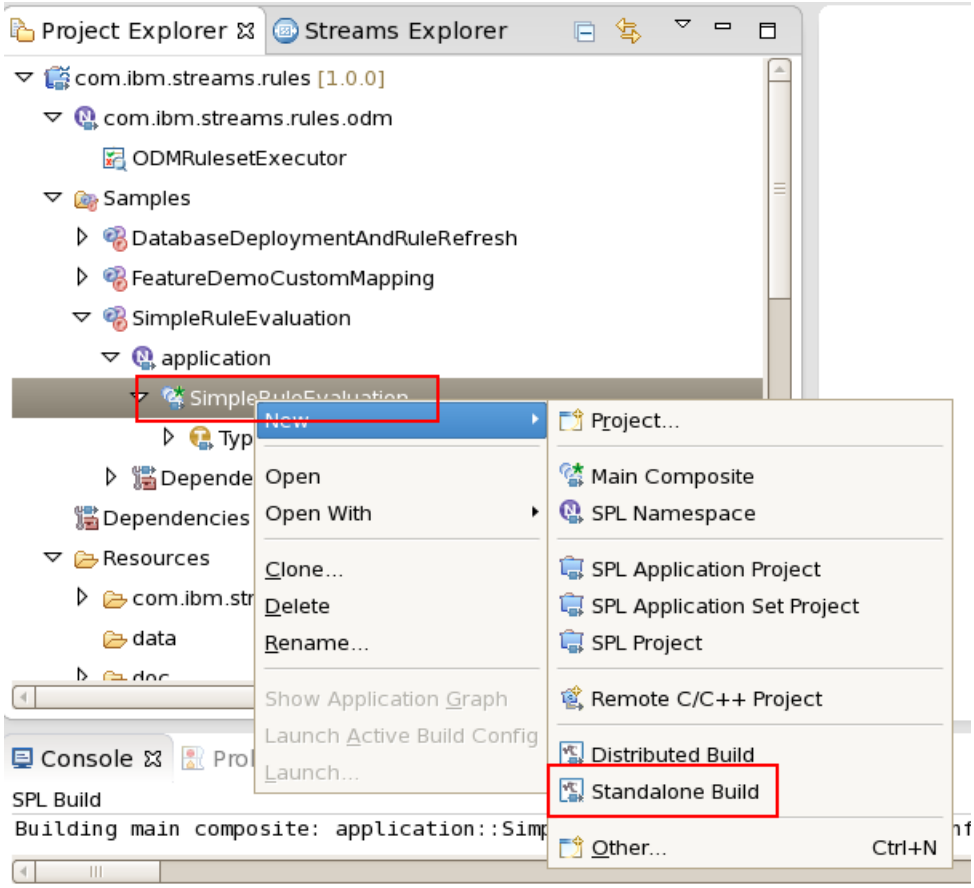
There is also an option to import individual toolkit SPL sample applications.

Once the project is imported into Streams Studio you can now build and run the sample applications.

The first step is to create a build configuration. This can be either Standalone meaning that it will run locally on a default stream or Distributed meaning that it can be deployed to a streams instance. In this tutorial we will create a standalone configuration.

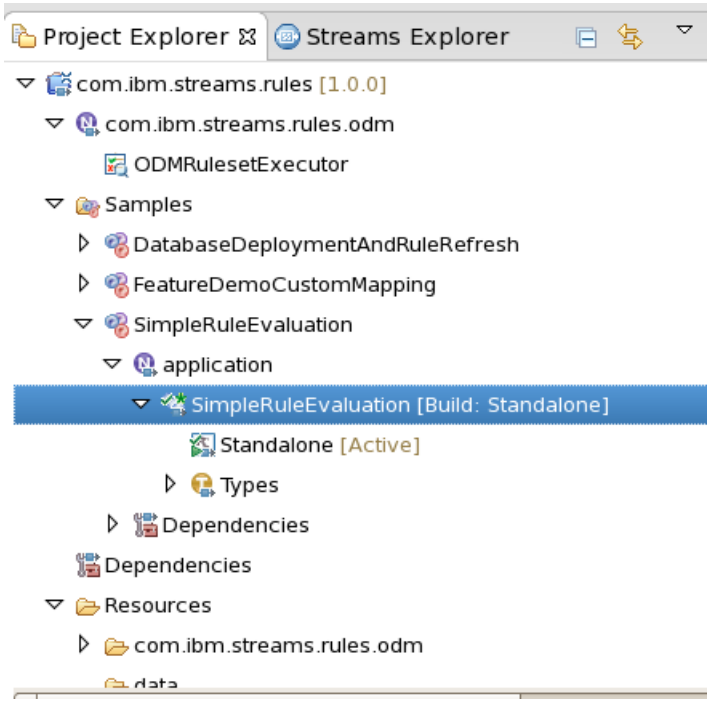
In the Project Explorer navigate to the sample you wish to run (SimpleRuleEvaluation) , under **application**, and right click **New>Standalone Build** as shown below in figure A.6.

Figure A.6 Creating a build configuration



This opens the Build configuration editor where further details may be modified if required. Click **OK** to close the editor and save the build configuration producing a **Standalone** build configuration as shown in figure A.7.

Figure A.7 Standalone build configuration

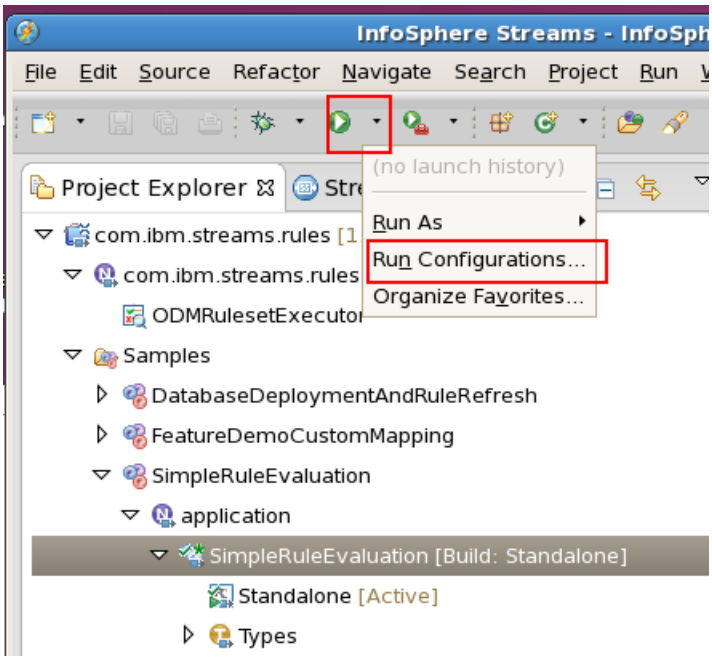


Now you have a build configuration you can establish a run configuration.

Build configurations that are imported may be read-only resources, and cannot be modified or synchronized. Right-click to delete the imported build configurations and re-create the Standalone build configuration.

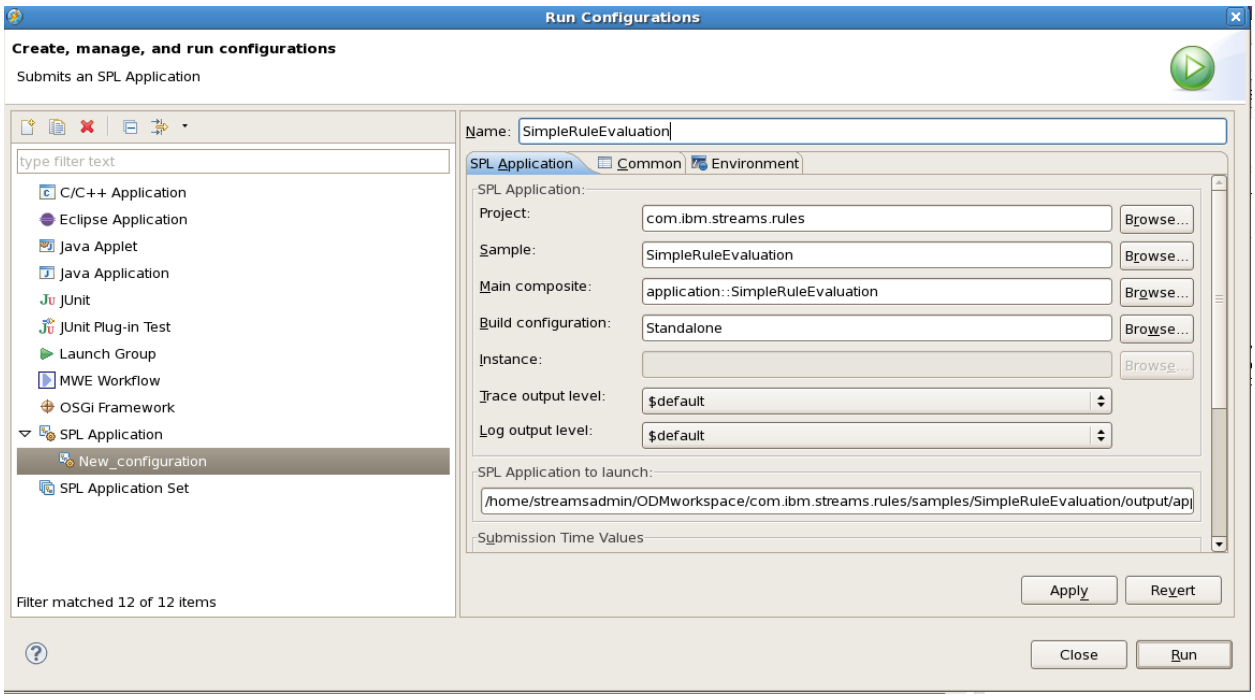
To create a new run configuration select the drop down next to the run icon and select **Run Configurations…** as shown in figure A.8.

Figure A.8 Creating a new run configuration



In the Run Configurations editor select SPL Application and right click New… to bring up the configuration editor. Change the fields to select the value for the sample (SimpleRuleEvaluation) as shown below in figure A.9.

Figure A.9 Run Configuration editor.

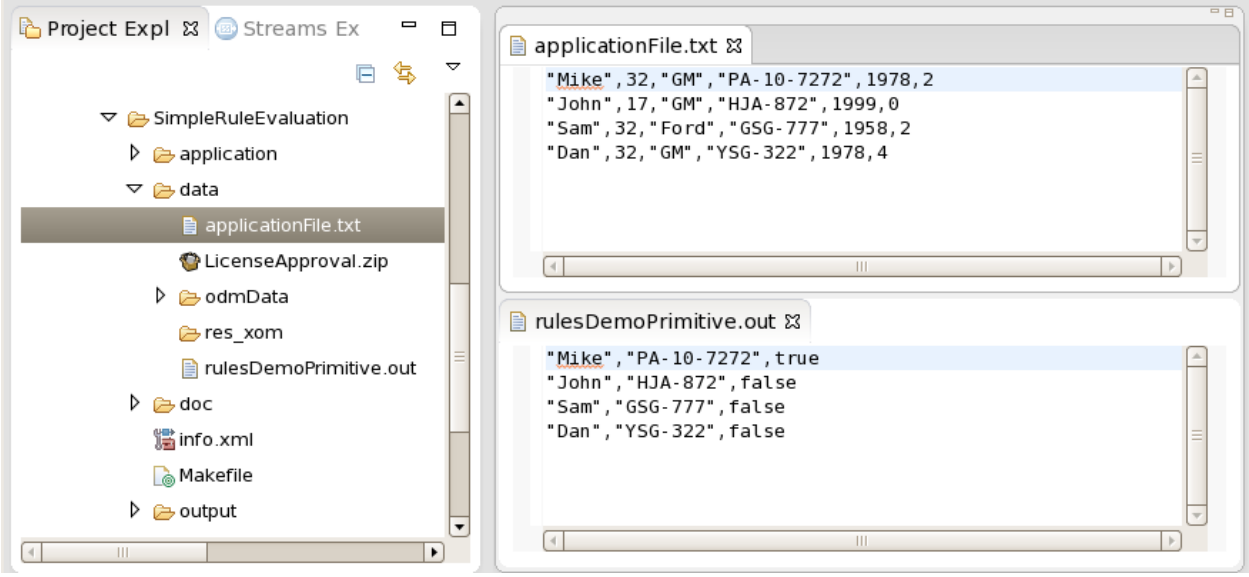


To save the run configuration click **Apply**.

You can then run this sample by clicking the **Run** button.

There is no output in the console from the SimpleRuleEvaluation sample but you can see the result of the execution by looking at the files used by the datasource node (applicationFile.txt) and produced in the data sink node (rulesDemoPrimitive.out) as shown in figure A.10.

Figure A.10 SimpleRuleEvaluation sample execution results



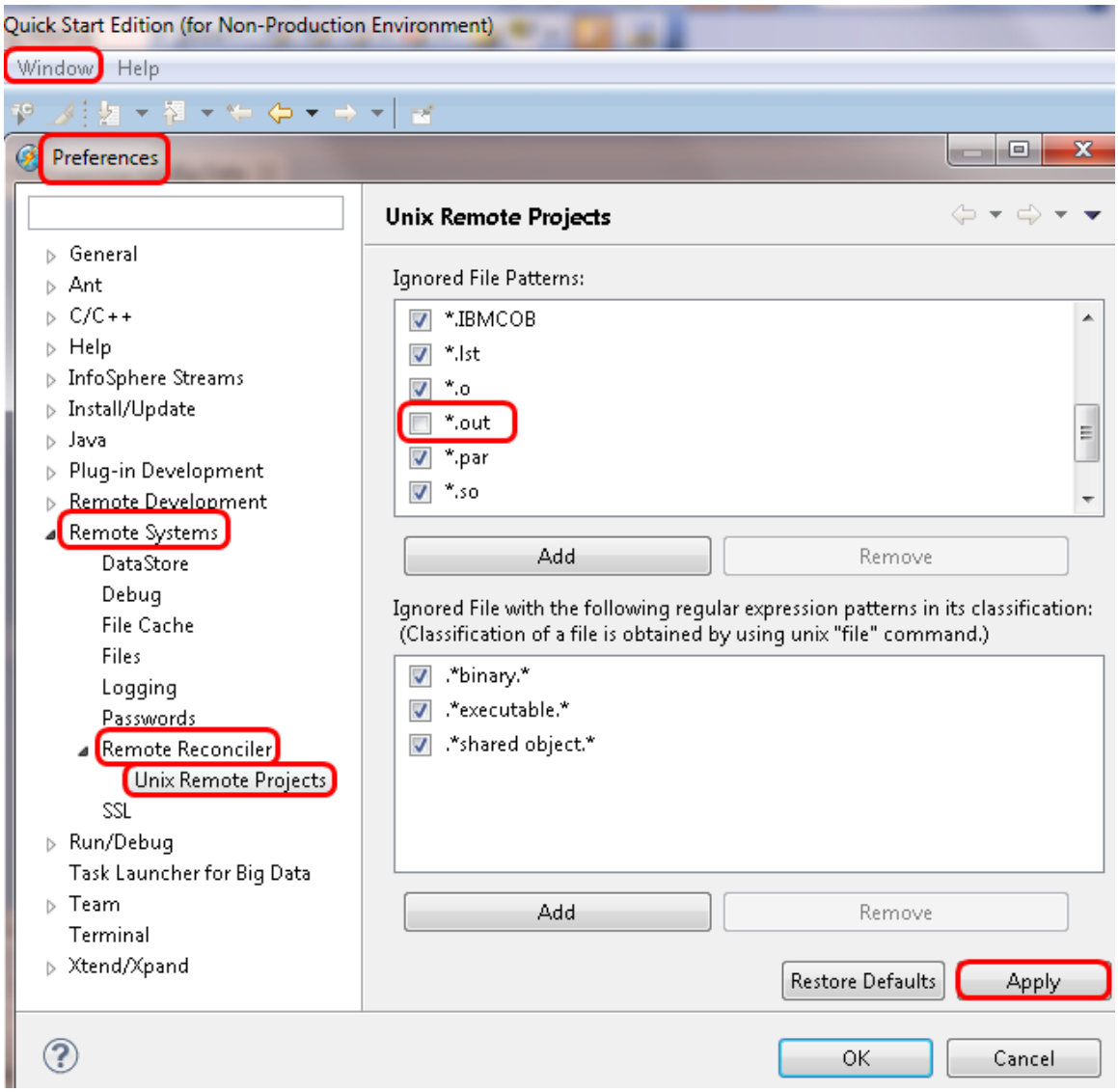
If you are working remotely with the Streams Studio, you need the Remote Reconciler to view the execution results. Right-click on **data** of the project and select **Remote Reconciler** where you have the options to synchronize, push, pull data or show data in remote reconciler/system.

The Remote Reconciler ignores some pre-defined file patterns, of which *.out is one. This means that you may not successfully synchronize the rulesDemoPrimitive.out file.

In order to change these Ignored File Patterns

- Select Windows > Preferences pnel
- Select Remote Systems > Remote Reconciler > Unix Remote Projects
- Deselect the suffix you wish to be synchronized as shown in the figure below.

Figure A.11 Ignored File Patterns in Unix Remote Projects



A similar approach can be applied to running the FeatureDemoCustomMapping application provided with the rules toolkit.

Running SPL Applications using Dynamically Deployed Rulesets

It is possible to configure the ODM Operator to use rulesets that are deployed to an existing Decision Server as long as that Decision Server:

- Is a version that is compatible with that used in the Streams ODM Operator
- has a network enabled database and
- has TCPIP notification enabled

The DatabaseDeploymentAndRuleRefresh sample is configured to operate in this way.

To use this sample you need to extract the provide rule project into Rule Designer and deploy it to Decision Server. The rule project used in this sample is in the project directory /DatabaseDeploymentAndRuleRefresh/data/ and is called MiniLoanRuleProject_ForDb_deployment_RuleRefresh.zip.

The rule project is based on a Java XOM (miniloanXom.jar) which is included in the rule project ZIP file and also in the /DatabaseDeploymentAndRuleRefresh/data directory. This jar has to be configured so that it is added to the streams classpath at runtime. As shown below in table x.x.

A RuleApp project can be created in Rule Designer and deployed to an existing Decision Server. The rulesetPath defined in the configuration parameters should correspond to this deployment. It is good practice to use a specific RuleApp version in the path to define the interface and XOM but to omit a ruleset version meaning that the operator will always use the latest ruleset deployed. You should not deploy the XOM for rule projects contained in the RuleApp as the miniloanXom.jar is already located in the classpath and the streams integration always uses file persistence for the XOM. Deploying the XOM will mean that the ruleset will have a reference that cannot be resolved by the local JSE RES and an exception will occur at runtime.

The Ruleset Executor Operator provides a number of parameters that can be used to configure its operation. In this sample you need to set up the Submission time values in the application configuration to reference the Decision Server and ruleset characteristics to which it should integrate. Typical values for a local Tomcat installation of Decision server running on a network Derby database (db2 values in brackets) are shown below.

Table A.1 Ruleset Executor configuration parameters

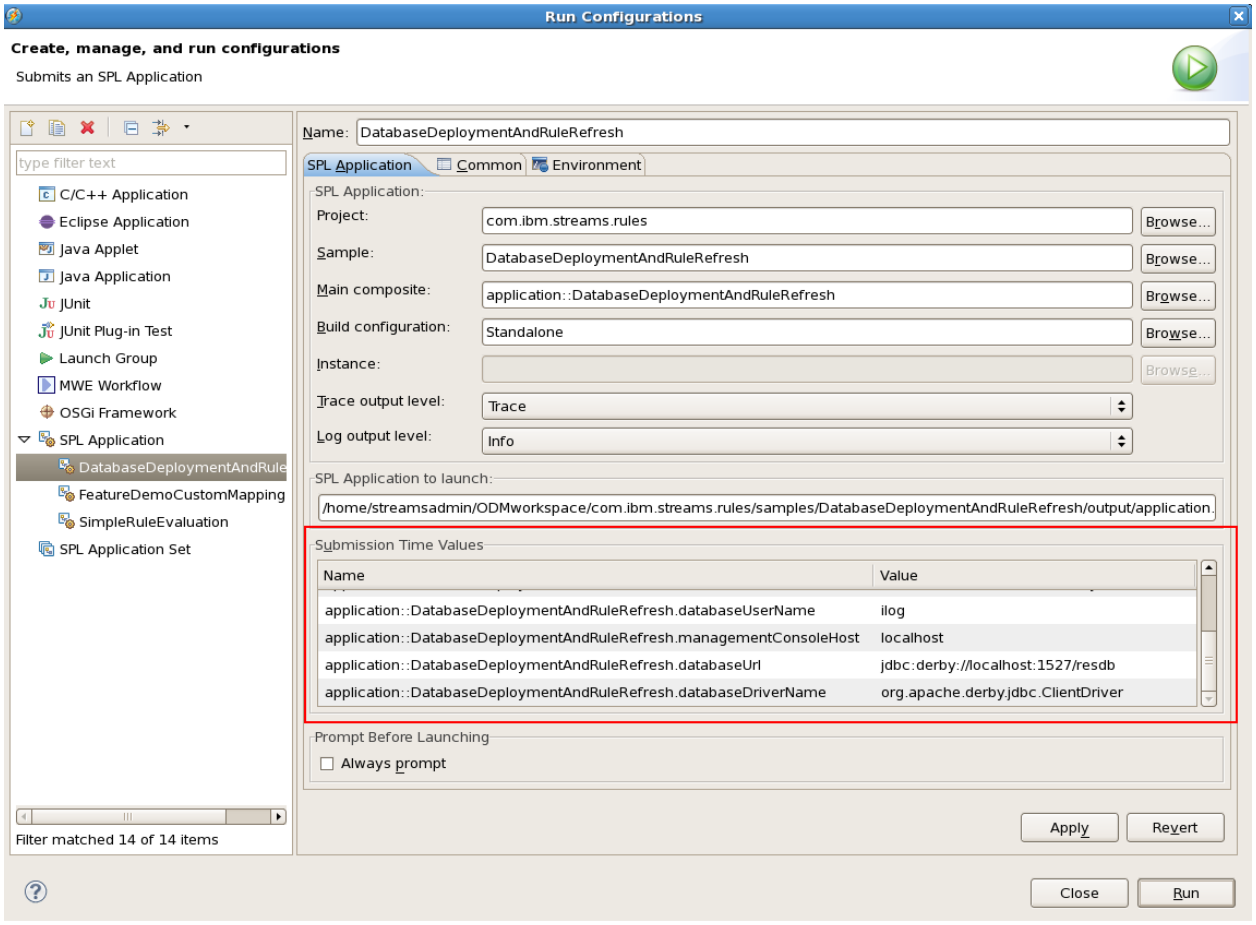
Parameter	Example Value	Comment
rulesetPath	/MiniloanRuleApp/1.0/MiniloanRules	Ruleset to be loaded from the decision server repository. The ruleset path does not include the ruleset version so that on notification of ruleset updates, the latest ruleset version is executed next.
managementConsoleHost	localhost	Management console host ip address or dns name used to register with the decision server.
managementConsolePort	1883	Port used to register for tcpip notifications of ruleset updates
databaseUrl	jdbc:derby://localhost:1527/resdb	Url of the decision server repository database
databaseUserName	ilog	Database user credentials

databaseUserPassword	ilog	Database user credentials
databaseDriverName	org.apache.derby.jdbc.ClientDriver	Jdbc client driver class name
databaseDriverPath	/home/streamsadmin/db-derby-10.10.1.1-bin/lib/derbyclient.jar	Classpath containing the client driver.

When these configurations are setup, the rulesets to be executed are taken from the Rule Execution server repository rather than the file system used in the previous samples.

A typical Run configuration is shown below in figure A.12.

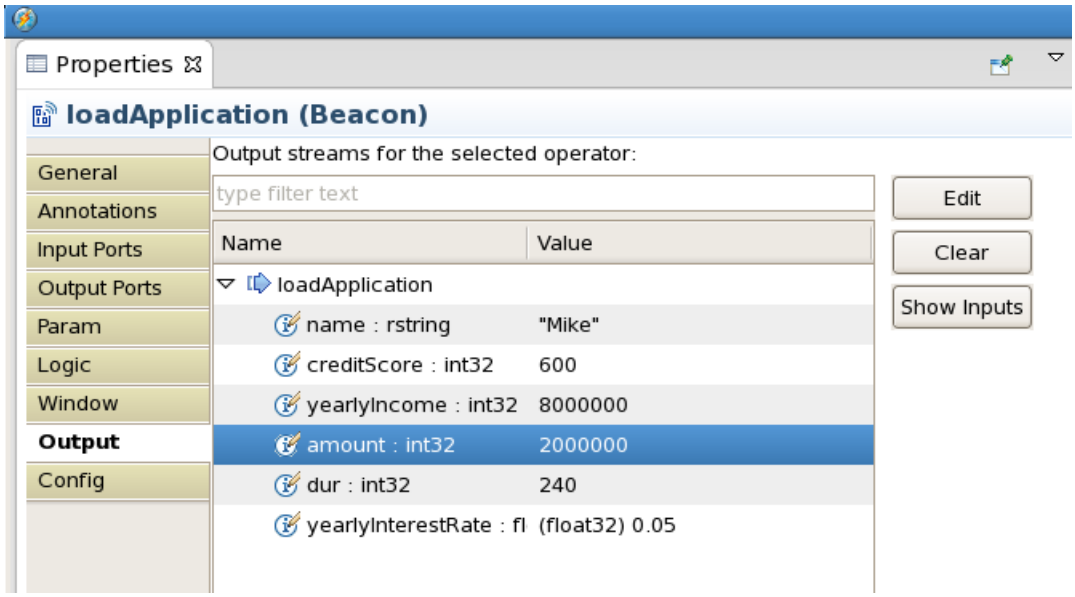
Figure A.12 ODM Operator node remote repository configuration



When executing streams with these settings the rules can be dynamically loaded into the operator when they are changed by deployment from Rule Designer or Decision Center.

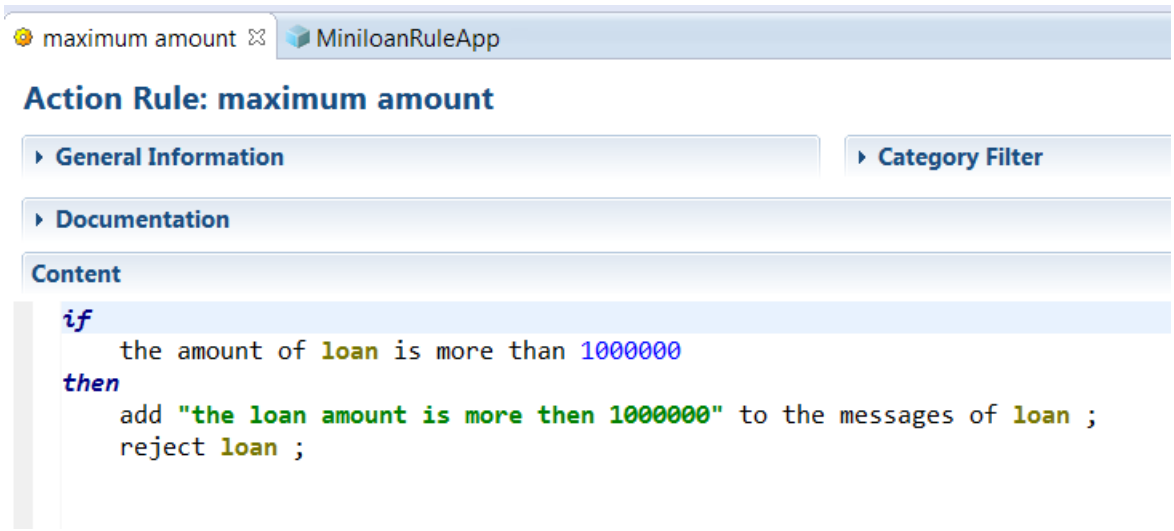
In this sample the input data to the rules is provide dthrough a Beacon node that sends the same set of data at regular intervals. This is configured to have a loan request amount of 2000000 as shown below in figure A.13.

Figure A.13 DatabaseDeploymentAndRuleRefresh sample Input source



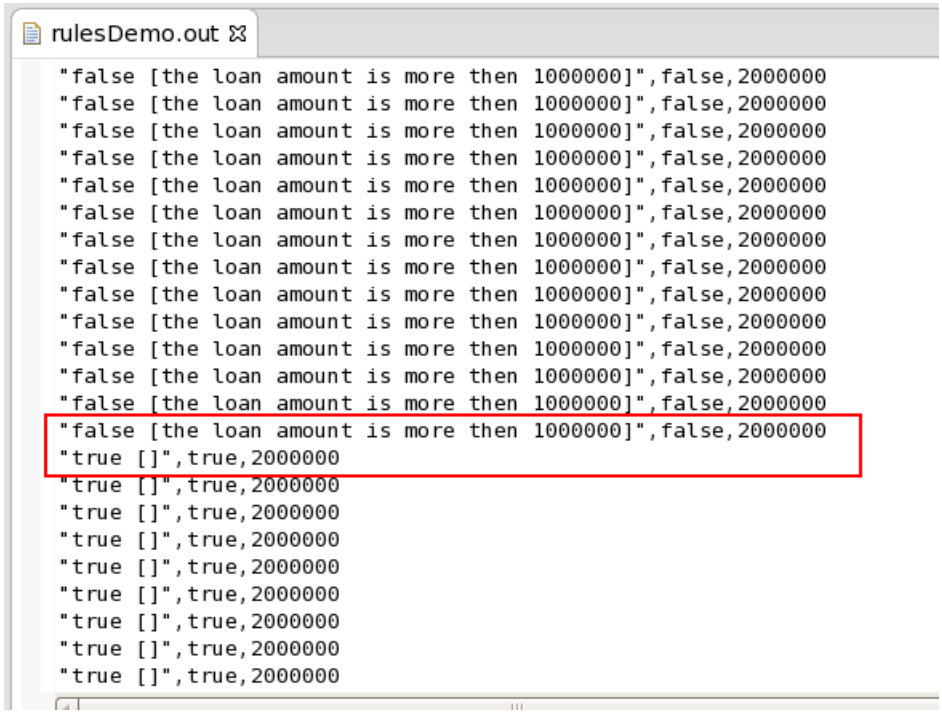
The default rules deployed with the sample include a rule that defines the loan application amount limit to be 1000000 as shown in figure A.14.

Figure A.14 Sample rule defining maximum loan amount



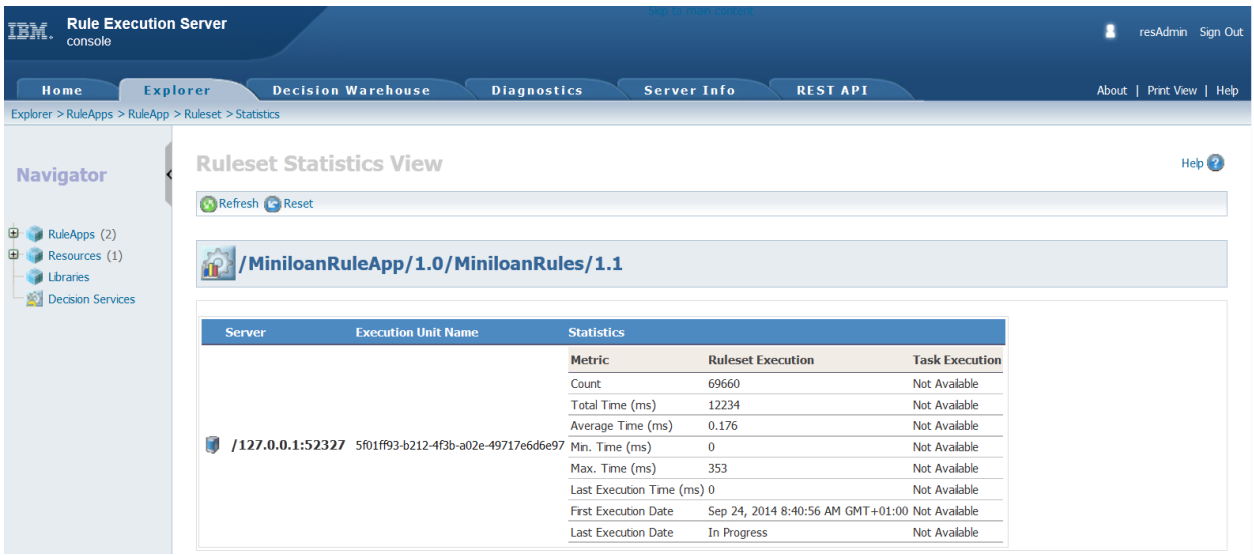
When the rules run in the stream each loan application should be rejected . If the amount is then changed to 5000000 and the ruleapp and ruleset redeployed, the decision changes to use the new limit and the loan is approved. This can be seen in figure A.15 below at the point of transition.

Figure A.15 Output from Data Sink as ruleset update is deployed



In addition the execution of rules can be managed and monitored in the Rule Execution Server Console. If tracing is enabled (this will incur a performance overhead) , statistics showing execution count and timings can be seen as shown in figure A.16.

Figure A.16 Ruleset execution statistics in Rule Execution Server console



Annex B: Configuring MQ and JMS for streams event handling on Linux

The use of the messaging toolkit within streams requires the installation and configuration of at least an MQ client and will usually require at least access to an MQ server in which the Queue Manager and Queues are running. This annex takes you through the key installation and configuration steps needed to support the scenario described in this article.

Installing MQ

The first step is to install MQ Server onto the Streams environment.

1. Open a command shell as root
2. Obtain and install MQ 7.5.0.1 or later with filename: WS_MQ_LIN_ON_X86-64_V7.5.0.1_EIM.tar.gz
3. Unzip into /opt/mqInstall or similar directory
4. Move to the install directory:
   cd /opt/mqinstall
5. Accept the licence:
   ./mqlicense.sh
6. Perform the install:
   rpm –ivh MQSeries*.rpm
7. This should install MQ into /opt/mqm

Configuring MQ with a Queue Manager

Once the installation is complete you should configure the mq administrative user and establish a queue manager.

1. Open a command shell as root.
2. Before issueing any commands ensure the MQ environment is setup
   . /opt/mqm/bin/setmqenv
3. Setup an mqadmin user:
   /usr/sbin/useradd –g mqm –d /home/mqadmin mqadmin
4. Set their password:
   passwd mqadmin (enter mqadmin password twice)
5. Change to the mqadmin user (enter password)
   su mqadmin
6. Create a queue manager:
   crtmqm –q IB9QMGR
7. Start the queue manager
   strmqm IB9QMGR
8. Start the tcpip listener
   runmqlsr –t tcpp –p 1414 –m IB9QMGR &
9. Run the MQSC console to configure the queue manager
   runmqsc IB9QMGR

10. Disable Channel Authorization
    ALTER QMGR CHLAUTH(DISABLED)
11. Enable automatic channel creation to allow use from MQ Explorer
    ALTER QMGR CHAD(ENABLED)
12. Open IBM WebSphere MQ Explorer
13. Select **Queue Managers,** right click and select **Show/Hide Queue Managers**
14. Click **Add**
15. Enter the Queue Manager name **IB9QMGR**
16. **Enter the following details:**
    Host Name: eg localhost
    Port: eg 1414
    Server connection channel: SYSTEM.AUTO.SVRCONN
17. Click **Finish**
18. The queue manager should now be visible in the navigator**.**
19. Create the queue to be used by selecting the **Queues** folder and clicking
    New-> Local Queue
20. In the name type the queue to be used: eg **BOOKEVENT_IN**
21. Accept defaults (or configure accordingly) and click **Finish**.

Configuring streams to recognize MQ

Once you have established your queue manager and queues, you need to need to ensure that the streams environment is configured to recognize them. In a production environment, the security would be configured by the WMQ administrator but in this example we will override security for simplicity.

For our JMSSink operator, we need to generate a .bindings file for JNDI lookup using a JMSAdmin.config file modified for our environment. The settings we use are:

- INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
- PROVIDER_URL=file:///opt/JNDI-Directory
- SECURITY_AUTHENTICATION=none

The steps below should be followed to configure streams to recognize MQ.

1. Open a command shell as root.
2. Ensure that the STREAMS_MESSAGING_WMQ_HOME environment variable is configured by including this export in the .bashrc file.
   export STREAMS_MESSAGING_WMQ_HOME=/opt/mqm
3. Create a directory to hold the MQ JNDI configuration. This will be needed by the streams JMS operators to find the JMS connections.
   mkdir /opt/JNDI-Directory
4. Copy the file JMSAdmin.config from opt/mqm/java/bin into this directory – this will hold details of queumanagers and should be edited to show:
   INITIAL_CONTEXT_FACTORY=
   com.sun.jndi.fscontext.RefFSContextFactory
   PROVIDER_URL=file:///opt/JNDI-Directory
   SECURITY_AUTHENTICATION=none

5. Issue the command:
   `/opt/IBM/mqm/V7.5/java/bin/JMSAdmin -v -cfg JMSAdmin.config`
6. In response to the prompts, define the QueueManager
   `DEF CF(ConnectionFactory) QMGR(IB9QMGR) TRANSPORT(CLIENT) HOSTNAME(localhost) PORT(1414)`
7. And define the queue:
   `DEF Q(BOOKEVENT_IN) QMGR(IB9QMGR) QU(BOOKEVENT_IN)`
8. On completion exit the Initial context and check that a .bindings file has been created in this directory.

The INITIAL_CONTEXT_FACTORY and PROVIDER_URL values in the JMSAdmin.config file are the values you need to set the initial_context and provider_url respectively in the connections.xml file for any JMSSinks. The provider URL directory is the directory where the .bindings file is generated with the WMQ JMSAdmin command.

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.