DECISION-ZONE

CYBER-PROTECTION

DZ AUDIT MANUAL

**To access the online version of this manual,
please scan the following QR code.**

Canadian Head Office
Decision-Zone Inc.
3 Willowbank Avenue,
Richmond Hill, ON L4E 3B4

Satellite US Office
2300 Clarendon Avenue,
Arlington, VA 22201

Info@decision-zone.com
www.decision-zone.com

Table of Contents

Preface

Welcome to DZ Audit. This document describes how to configure and use the product to perform real-time or historical business process event audits.

How this Guide is organized

This document contains the following sections:

Chapter 1, "Introduction"

Chapter 2, "Configuration"

Chapter 3, "Runtime"

Chapter 4, "Tools"

Chapter 5, "Event Providers"

Chapter 6, "Additional features"

Conventions

This manual uses the following conventions:

- Monospace
  - Specifies file names, object names, and programming code.

- Italics
  - Identifies a variable
  - Indicates a value that you must enter
  - Introduces new terminology, highlights and provides emphasis

- Bold
  - Highlights items and indicates specific items in a graphical user interface.

Introduction

Overview

The DZ Audit performs real-time or batch business process event audit. Assuming that the organization has the business processes documented in a state-chart diagram format, the product can import such business process definitions in XMI format and provide a runtime layer that correlates events and highlights any anomaly in the process execution. Additionally, the resulted sets of correlated events can be further filtered by using pattern expressions. The resulted filtered event sets are considered 'complex business events'.

Concepts

To perform a business process event audit, the 'DZ Audit' requires several configuration files to be present.

The most important input file is the business process definition file in UML state chart XMI format. If none is available, 'DZ Audit' can assist the user to import or create a new business process XMI file, by using the built in diagram editor.

A mapping file will describe the link between the events associated with transitions in the business process definition and real data from real data-sources. Two main types of events are supported: real-time and historical.

The events are extracted from the real world systems by using 'Event providers'.

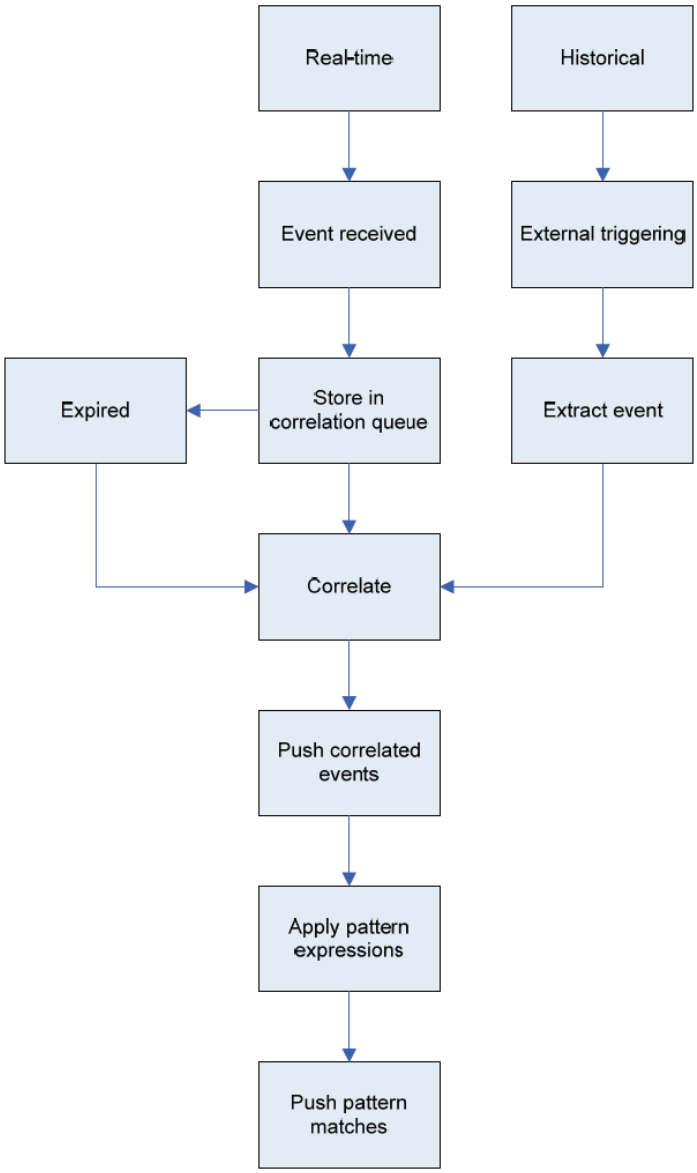
These providers are of two main types:

- Real-time. Can handle expiration/jitter aspects of the real-time systems. (An event will be 'expired' if it is not correlated after a configurable predefined period of time. Since the sequence in which the events are picked up from the middleware layers cannot be

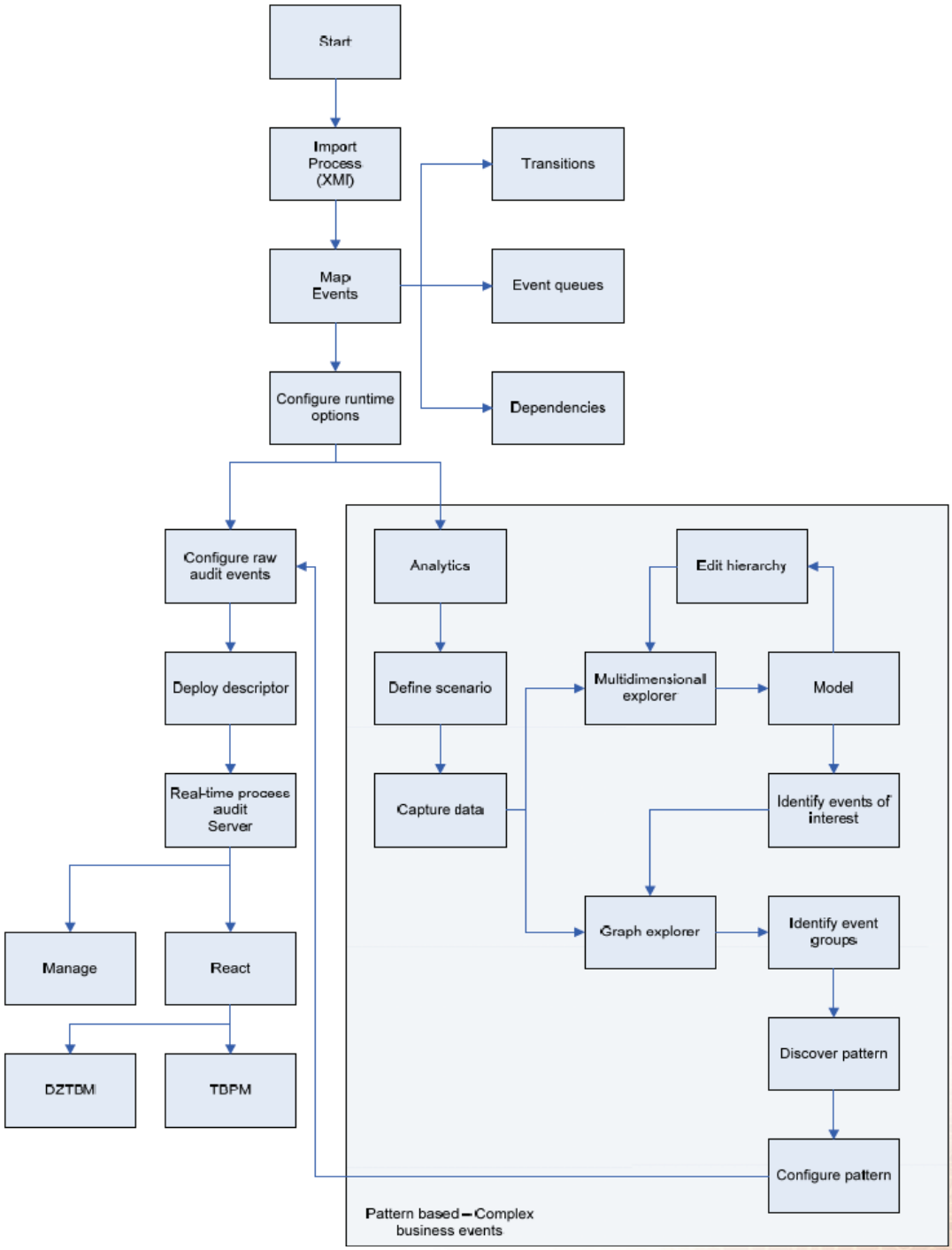
guaranteed to be in order, jitter parameter allows the correlation process to reorder the events.)

- Historical. Extracts events from databases or other historical sources. If one of the events supplied by this provider type is connected to a terminal state in the process definition, external triggering of the correlation process is required. However, if used in the same process audit scenario, the real-time events will still be subject to expiration/ jitter constraints.

The following diagram describes the sequence of operations performed in a runtime scenario:



The following picture describes the typical steps followed while configuring a process audit:

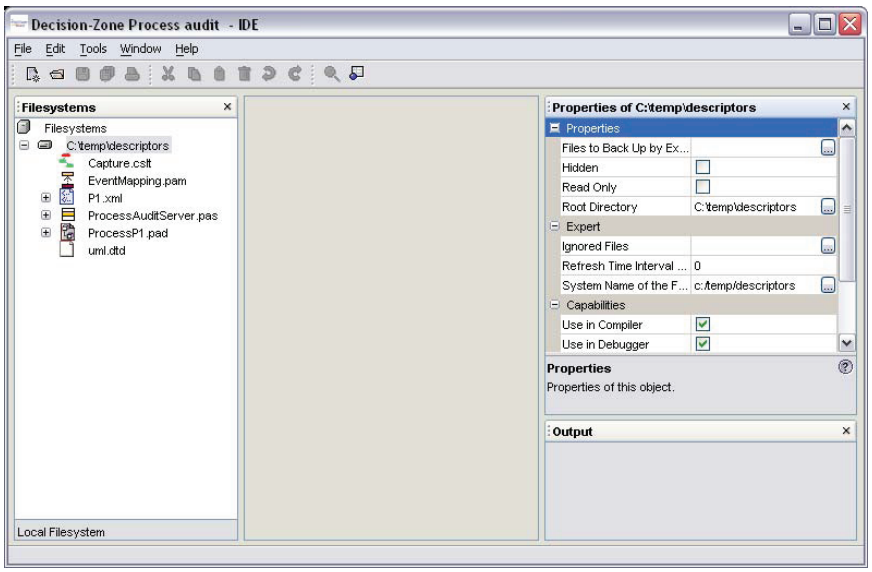


Configuration

Startup of the IDE

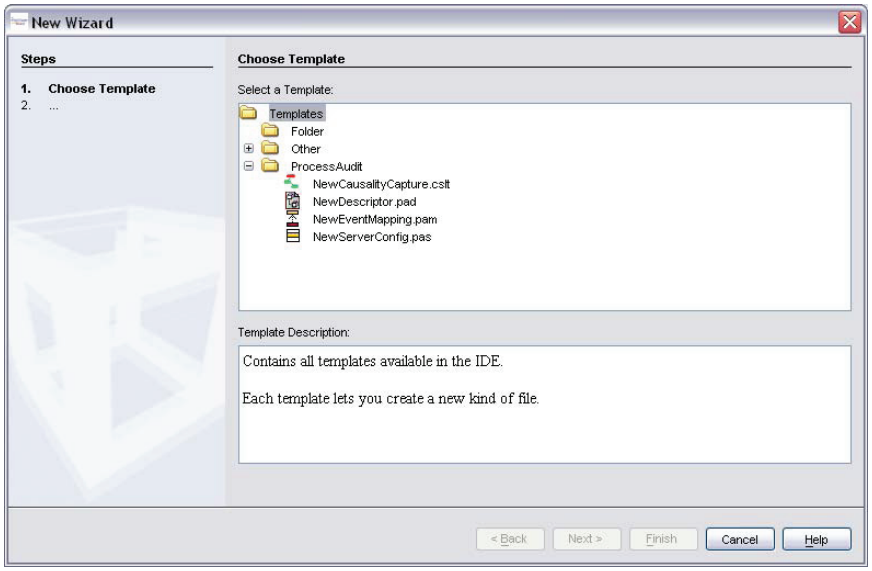
The IDE can be started by calling 'startIDE.bat' or the appropriate shell for your operating system. After startup, the IDE 'remembers' the last mounted file-systems. To mount a new one, select 'File' -> 'Mount File-system' ->…

After startup, the screen should look like this:



Create new resource

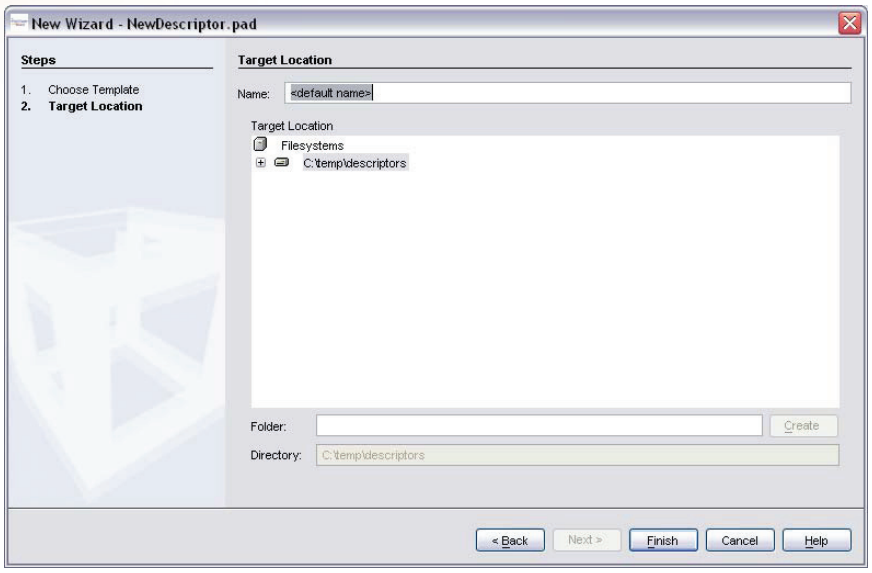
Typical usage scenarios require the creation/manipulation of a set of files. A set of file templates are available and the associated 'wizard' can be invoked from 'File'->'New' menu or by selecting 'New' from the context menu associated with a directory.



The following templates are available:

- NewDescriptor.pad – Template for a new descriptor.
- NewEventMapping.pam – Template for event mapping to XMI definitions.
- NewServerConfig.pas – Template for server configuration.
- NewCausalityCapture.cslt – Template for a new event capture file.

After a template type has been selected, the wizard screens will prompt the user to select the target location of the newly created file. The name should not include the extension of the file. This will be added automatically.



Descriptor

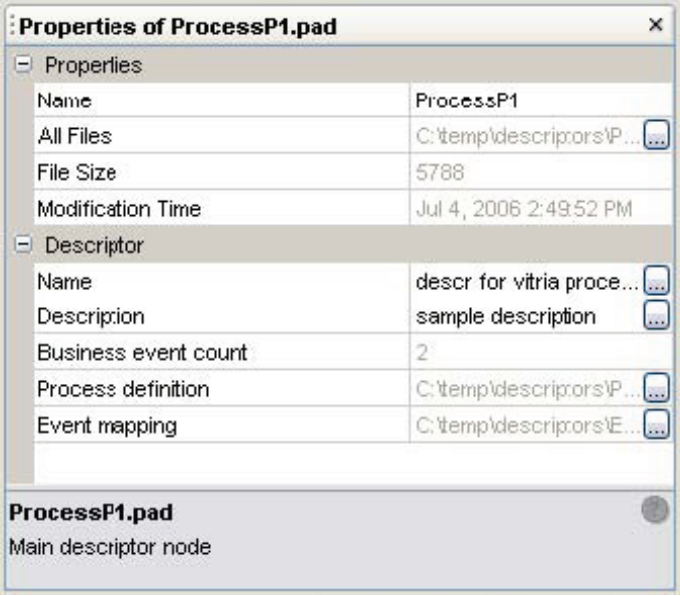
A descriptor will contain all the information or locations of information required for a business process event audit. The default extension of a descriptor file is 'pad' (Process Audit Descriptor).

IMPORTANT: *After a new descriptor has been created from a template, a default un-configured business event will be created. The topic and the provider name will be required to be filled in before the descriptor will be deployed in a server for execution.*

After making a change of the descriptor's properties or the properties of one of its children nodes, right click on the descriptor node and select 'Save' to persist changes.

By selecting the node, the following properties are made available.

Property	Description	Default
Name	Name of the descriptor	newDescriptor
Description	Short description	
Business event count	Number of business event definitions (Read-only)	1
Process definition	Location of the process definition (XMI) (Read-only)	NA
Event mapping	Location of the event mapping file (Read-only)	NA



A descriptor always has three sections:

- Process definition
- Business events
- Analytics

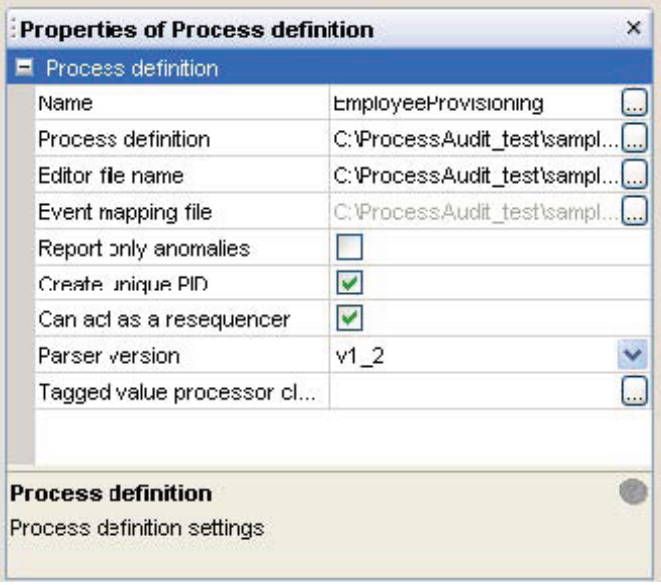
Descriptor – Process definition

This section is mandatory. It contains the reference to the XMI file and to the mapping file.

By selecting the node, the following properties are made available.

Property	Description	Default
Name	Business name of the process.	newProcessDefinition
XMI file name	Location of the process definition (XMI).	NA
Editor file name	If the process was created or edited with the supplied UML tool, then this field points to the location of the tool specific project file.	Temporary file
Event mapping file	Location of the event mapping file (Read-only).	NA
Report only anomalies	Boolean property. If true, only the anomalies will be published.	false
Create unique PID	Boolean property. If true, a unique identifier will be assigned to each correlated event set. Consult Rapide documentation for additional information.	true
Can act as a resenquencer	If the mapping resequence properties are properly populated the server will resequence (arrange in proper order) the	

	transition triggering events	
Parser version	Choice. Select the version of the parser to match the version of the business process XMI file.	V1_0
Tagged value processor class name	Class name that implements the interface 'TaggedValueProcessor'. Consult the programmer guide for additional information.	



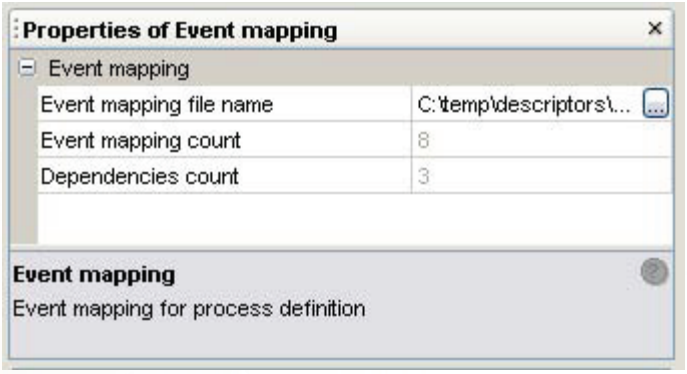
If the XMI file name is invalid or it points to an invalid file or a directory, a red dot will appear beside the 'Process definition' node.

If the process definition was edited or created with the built-in UML editor tool, to edit the project file associated with it, select 'Edit process definition' from the context menu associated with 'Process definition' node.

Descriptor – Event mapping

This section is mandatory. It contains the reference to the event mapping file and it allows the user to update the mapping information. By selecting the node, the following properties are made available.

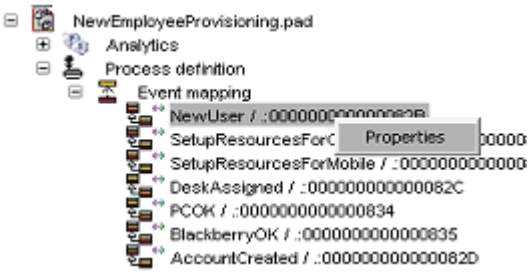
Property	Description	Default
Event mapping file name	Location of the event mapping file name	NA
Event mapping count	Number of defined event mappings	0
Dependencies count	Number of listed dependencies	0



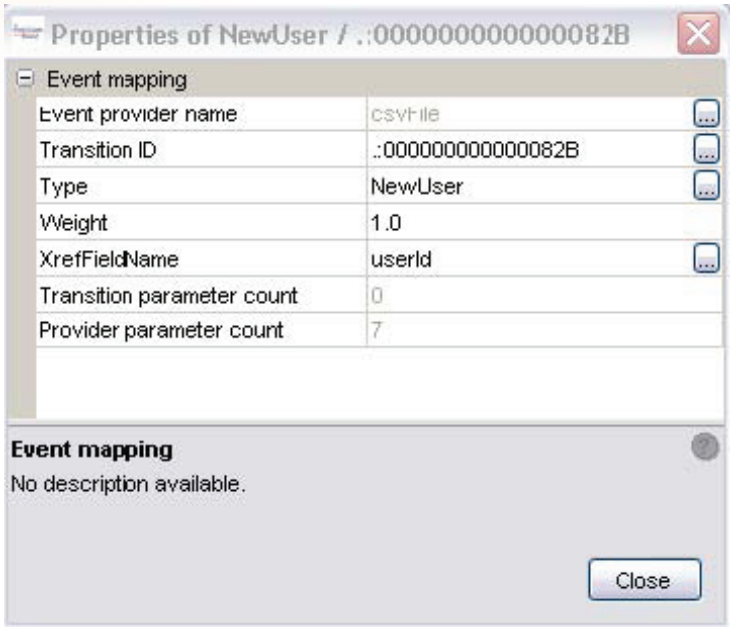
Set mapping reference

If the specified event mapping file name is invalid or it points to an invalid file or a directory, a red dot will appear beside the 'Event mapping' node.

Immediately after a valid mapping file was specified, the 'Event mapping' node will display a set of child nodes representing the declared mappings from the configuration file.

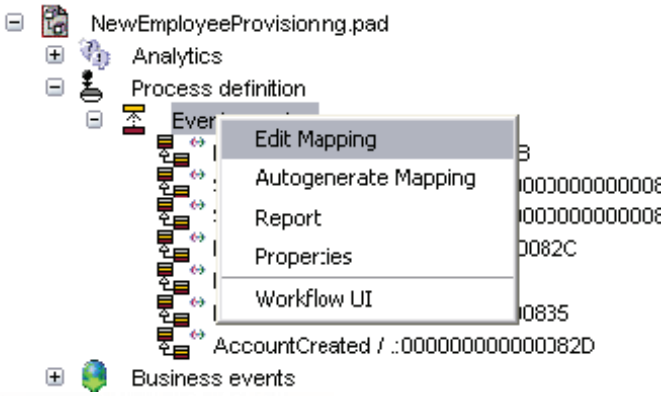


Property	Description	Default
Event provider name	Name of the configured event provider.	
Transition ID	ID of the transition as declared in XMI.	
Type	Class name of the event type as declared in XMI.	
Weight	Double between 0 and 1 representing the probability of the event. 1 is the highest.	1
Xref Field Name	Name of the session correlation field. Optional.	
Transition parameter count	Count of the parameters of the mapping (identifiers declared in the guard condition associated with the transition).	
Provider parameter count	Count of the parameters of the provider.	



Edit Mapping Wizard

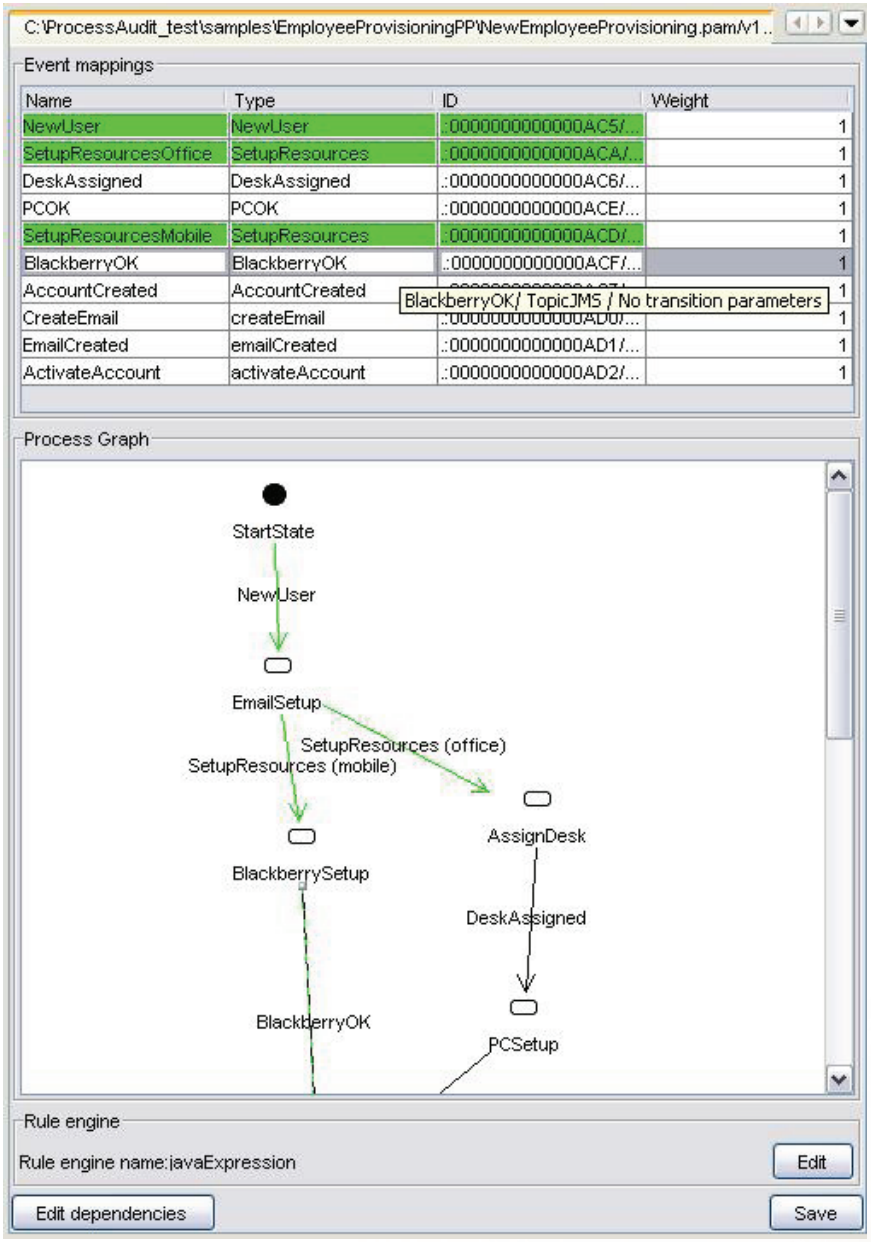
To edit the mapping file, select 'Edit Mapping' from the context menu associated with the node 'Event Mapping'.



The event mapping file contains two main panels:

- Event mappings – actual mapping.
- Process Graph – graph of the process.

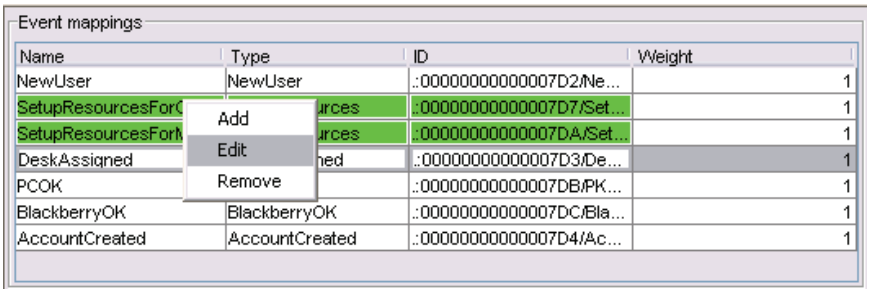
Mapping tool screen:



The event mappings listed are color coded:

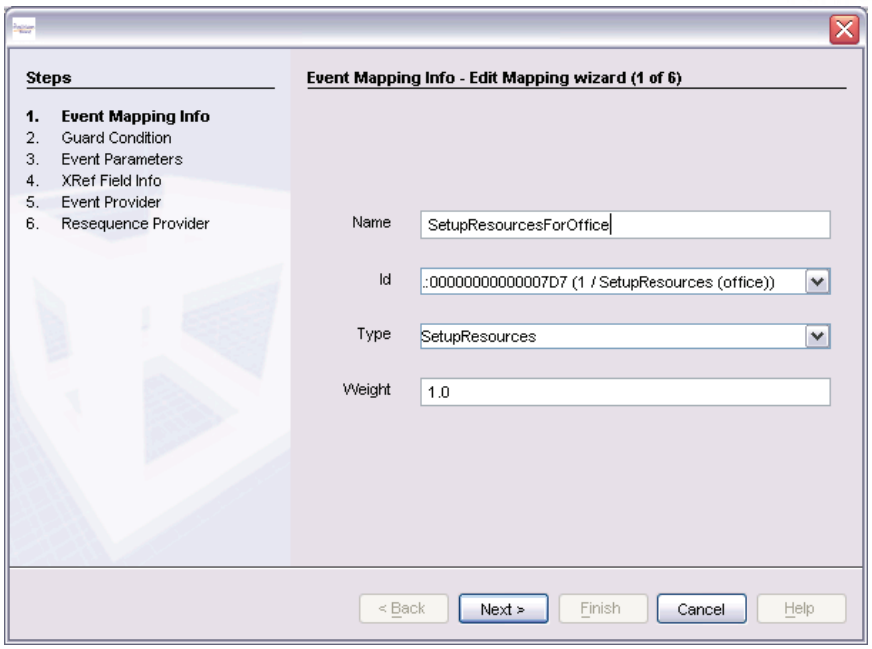
- Green: the mapping has a guard condition associated with it and a provider configured.
- Yellow: the mapping does not have a provider associated with it (virtual event).
- Red: the mapping has a guard condition declared in XMI but no event provider associated with it. This represents a fault condition.

To add/edit/remove an event mapping, select the corresponding action from the menu associated with a right click on 'Event mappings' table/rows.



To edit the mapping file, a wizard will take the user through the following steps:

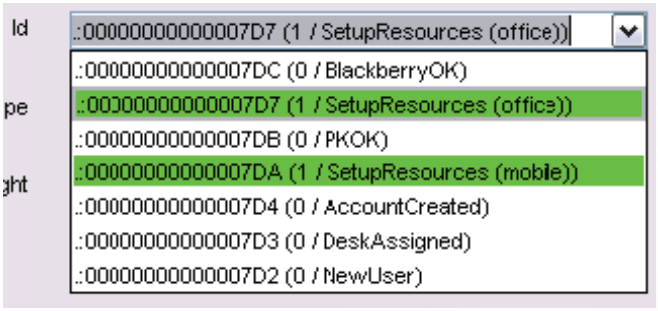
1. 'Event Mapping Info' panel



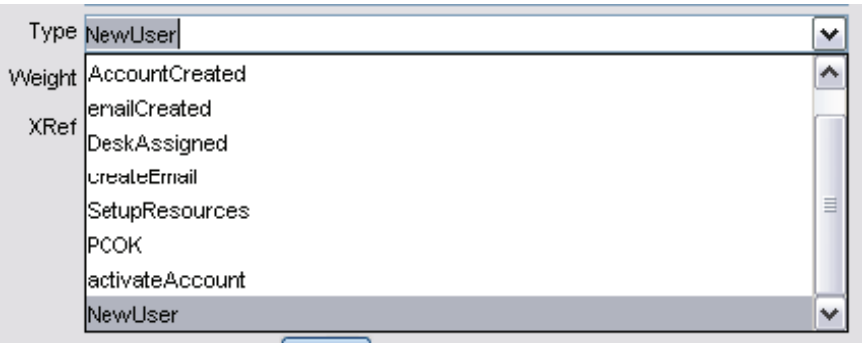
Name: the declared name of this event mapping.

Id: select from a list of event id's as declared in XMI. The drop down list will mark with a green background color all the transitions that have a guard condition associated. To overwrite, type an ID.

After an ID selection, the type field will be updated with a matching type for the selected transition.

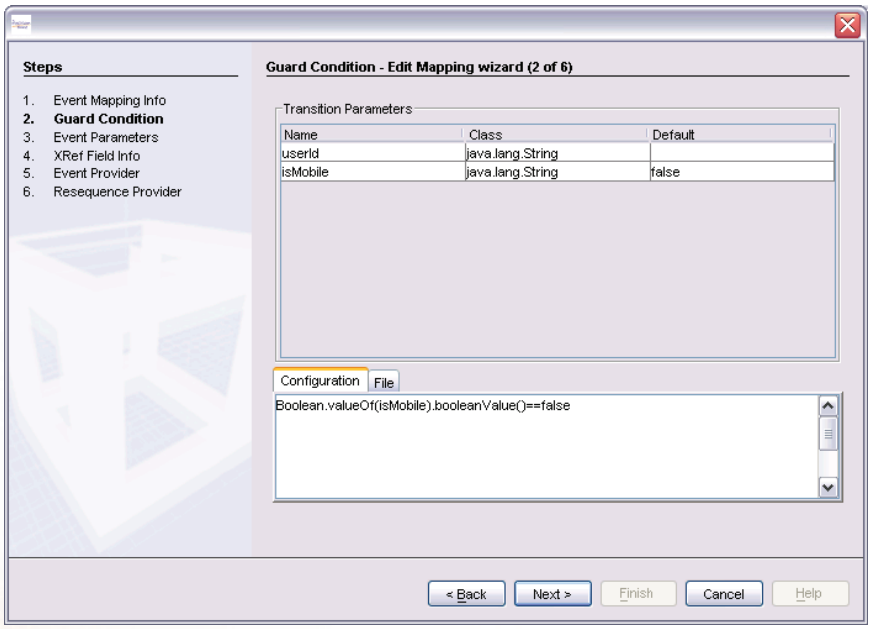


Type: select from a list of declared event types. To overwrite, type an event type as a java class name.



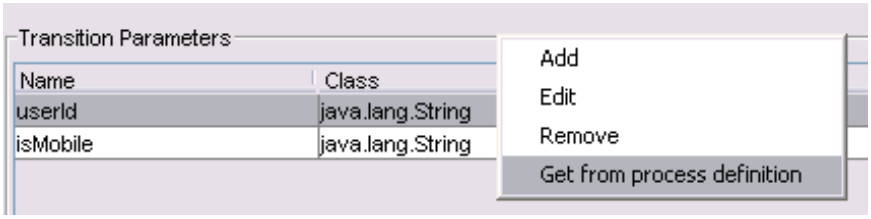
Weight: Double between 0 and 1 representing the probability of the event. 1 is the highest.

2. 'Guard Condition' panel



Guard transition parameters are initially populated with the guard condition parameters defined in the process definition file.

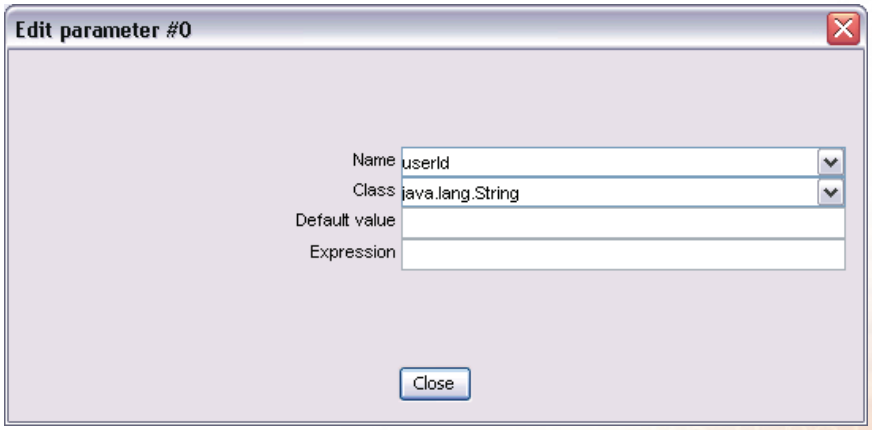
To add/edit/remove or 'Get from process definition', select the corresponding action from the context menu associated with the table/rows.



Attributes:

- Name: drop down list. Select the required parameter (identifier). Its name is extracted from the guard condition expression.
- Class: drop down list. The content is retrieved from the 'lookup.xml' file under /lookup/parameterTypes section. Type a valid java class name to overwrite.
- Default value: if the field is not found or cannot be extracted from the captured event at runtime, this will be the value presented to the evaluation layer.
- Expression: expression for calculated parameters.

Edit transition parameter screen:

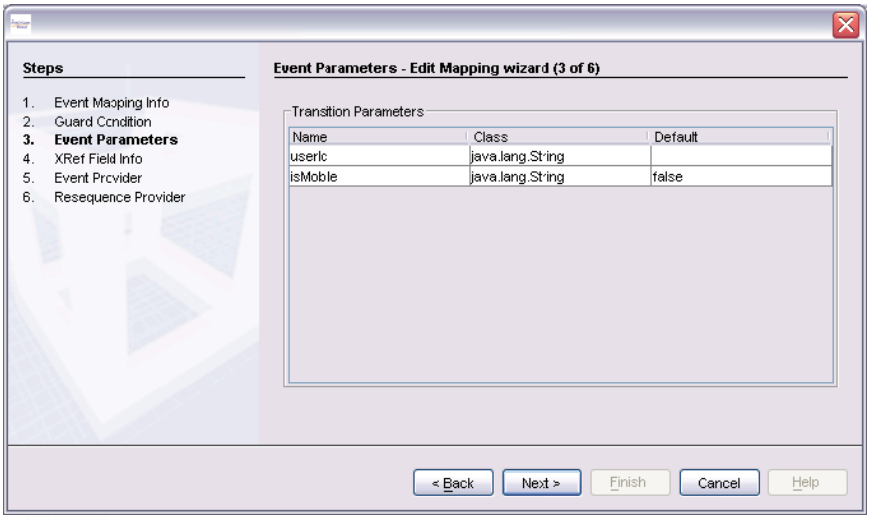


The guard condition source code is located on the bottom side of the screen. If the current rule engine is configured to use the configuration as a source for the guard code, the 'Configuration' tab becomes active and user can change the guard condition code. If the rule engine is configured to use a file as a source for the guard condition, the 'File' tab is active and the content of the file is shown (read only).

Initially the 'Configuration' tab is populated with the guard condition extracted from the process definition.

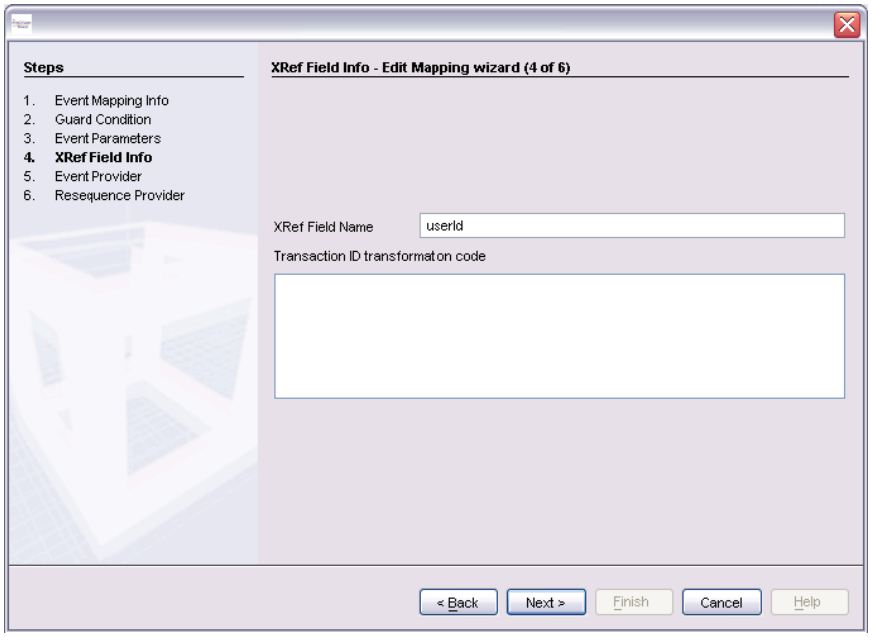
If the selected rule engine is "javaExpression" the guard condition result type must be boolean.

3. 'Event Parameters' panel



This panel allows editing expressions for calculated event parameters. Event parameters are identical to transition parameters presented in the 'Guard Condition' panel.

4. 'XRef Field Info' panel

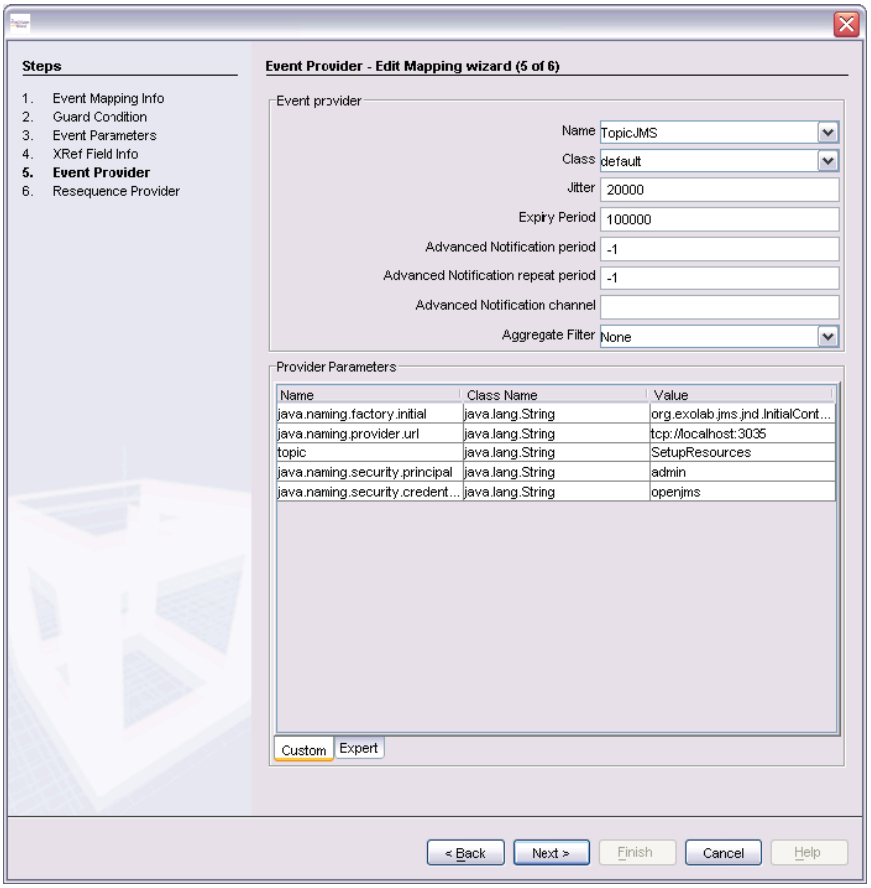


XRef: select one of the event fields used to uniquely identify a transaction. If the value of the next transition xRef field represents the result of a computation, enter the transformation code in the text area provided.

```
System.out.println("currentTransactionId: "+currentTransactionId);
nextTransactionId = "j"+currentTransactionId;
System.out.println("nextTransactionId: "+nextTransactionId);
```

On the translation code 'currentTransaction' represents the value of the xRef field from the current transition and 'nextTransactionId' represents the expected value of the next transition xRef field.

5. 'Event Provider' panel



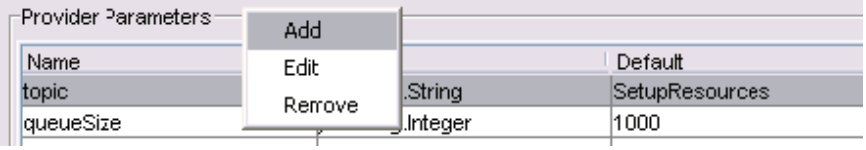
The upper panel contains the common parameters of the most important event provider. These parameters are:

- Name: the provider's name.
- Class: auto populated field, represents the class name of the provider.
- Jitter: delay compensation value.

- Expiry Period: period of time to keep the events in the correlation queue. After an event was marked as 'expired' the correlation process will, most likely, tag it as an anomaly. The runtime can generate warings related to delays in event sequences. Multiple warning types can be sent to the same recipient.
- Advanced notification period: the time period before the engine starts sending advance notifications. The runtime can generate warnings related to delays in event sequences.
- Advanced notification repeat period: the time interval between notifications.
- Advance notification channel: advance notification channel name.
- Aggregate filter: drop down field with the following possible values:

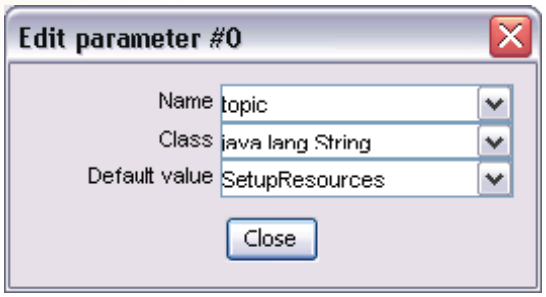
  • None
  • Repetitive, allowing repetitive events.
  • Retry, allows ignoring retries.

To add/edit/remove a parameter for the event provider, select the required action from the menu associated with right click on 'Parameters' table/rows.



This screen allows the editing of the provider parameter.

- Name: drop-down list. The provider parameters are declared in 'lookup.xml'. Type a new name to overwrite.
- Class: type of the parameter (java class name).
- Default value: the value to be passed to the event provider at startup.



Generic parameters (real-time providers):

Property	Description	Default
topic	Topic on which the even provider will listen for events.	Mandatory
queueSize	The size of the event queue (number of events to be stored in the correlation queue before they are marked as 'expired'). Real time providers only.	10
expiryPeriod	Period of time to keep the events in the correlation queue. After an event was marked as 'expired' the correlation process will, most likely, tag it as an anomaly.	20000
eventQueueClassName	Implementation class name of the correlation queue. Fully qualified java class name. The implementing class should implement 'com.decisionzone.pattern.Queue' or 'com.decisionzone.pattern.ExpiryQueue' interface.	com.decisionzone. pattern.xmi.engine. EventQueue

Repetitive parameters:

Property	Description	Default
repetitiveDelay	The interval of time in which the events are considered in the same repetitive sequence.	1000
repetitiveMaxEventCount	The number of events that should be received in the configured interval of time, that will trigger the transition.	1
repetitiveQueueSize	The size of the event queue (number of events to be stored in the repetitive queue).	10

Retry parameters:

Property	Description	Default
retryDelay	The interval of time in which the events are considered in the same retrying sequence.	1000
retryMaxEventCount	The maximum number of retries.	1
retryQueueSize	The size of the event queue (number of events to be stored in the repetitive queue).	10

Validation parameters:

Property	Description	Default
validatorClassName	The validation class name. Must implement `the EventValidator interface.`	1000

validatorFieldNames	The event field names used in validation	
validatorContinueCorrelation	If true and the validation failed, the engine continue the correlation of events	
validatorIgnoreErrors	If false the and the validation failed, the validator will send the event to the engine. If true only a reply message containing the error message, will be send to the requestor, the engine will ignore this event.	

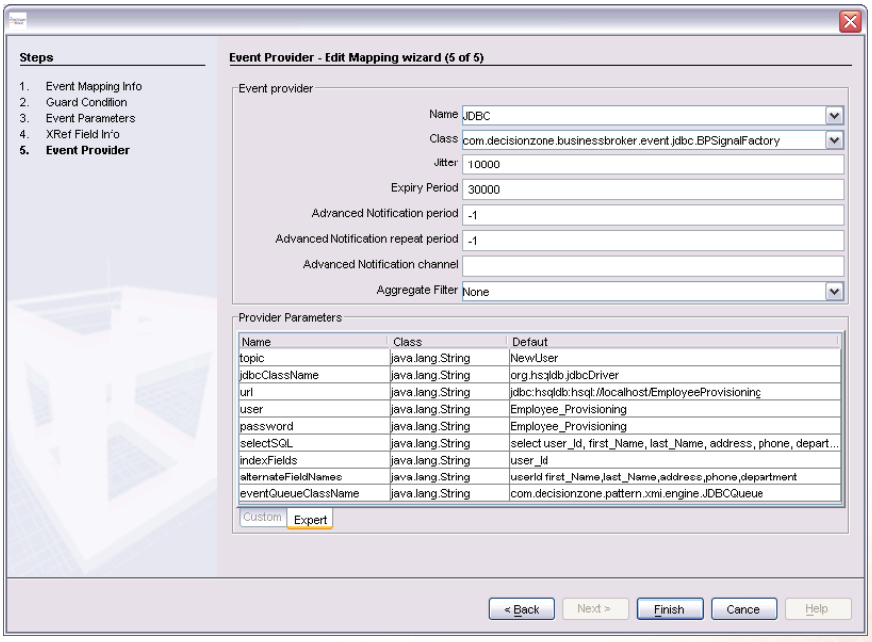
Additional parameters for historical event provider:
Provider Name: JDBC
Provider class: com.decisionzone.businessbroker.event.jdbc.
BPSignalFactory
Valid only if eventQueueClassName= com.decisionzone.pattern.xmi.
engine.JDBCQueue)

Property	Description	Default
jdbcClassName	jdbc driver class name. Valid only if eventQueueClassName= com.decisionzone.pattern.xmi.engine.JDBCQueue	oracle.jdbc.driver.OracleDriver
url	Database server connection URL	For an Oracle database: jdbc:oracle:thin:@<host>:1521:<sid>
user	Database user name	
password	User password	
selectSQL	A valid select sql. The select sql returned columns represent the event fields. Sample: "SELECT CTXID, TYPE, PERFORMANCEINDICATOR FROM event1" will create events having CTXID, TYPE, PERFORMANCEINDICATOR as fields,	
	each row of the table event1 representing an event.	
indexFields	Comma separated list of fields that uniquely identify a result row ("primary key" or "unique key"). Sample: event1 is uniquely identify by CTXID.	
alternateFieldNames	A comma separated list of field names that will replace the automatically generated field names (column names returned by the select SQL).	

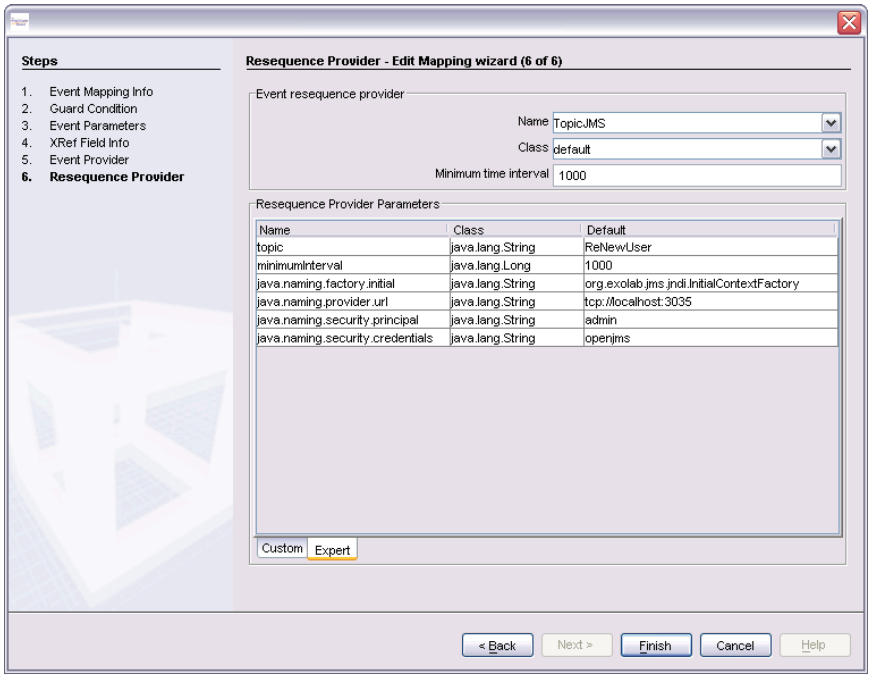
Additional parameters for rule engine validator:

Property	Description	Default
ruleEngineConfigFile	The rule engine xml configuration file	
ruleEngineURI	The URI of the desire rule engine	
ruleExecutionSetURI	The URI of the desire rule engine execution set	
ruleEngineDbConnectionDriver	Database connection driver class name (optional)	
ruleEngineDbConnectionURI	Database connection string (optional)	
ruleEngineDbConnectionUser	Database user name (optional)	
ruleEngineDbConnectionPassword	Database user password (optional)	
ruleEngineDbQuery_	SQL queries prefix (optional)	

Sample configuration screen:



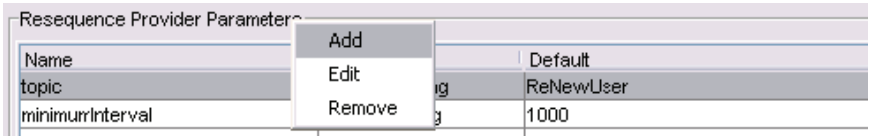
6. 'Resequence Provider' panel



The upper panel contains the common parameters of the most important event provider. These parameters are:

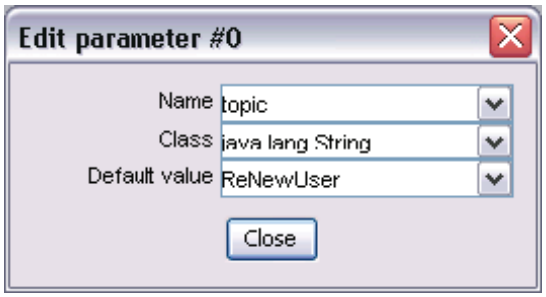
- Name: the provider's name.
- Class: auto populated field, represents the class name of the provider.
- Minimum time interval between the resequence event. In order to assure the correct order of the events, the resequence engine can delay the events to respect a minimum time interval between subsequent events.

To add/edit/remove a parameter for the event provider, select the required action from the menu associated with right click on 'Parameters' table/rows.



This screen allows editing of the parameters.

- Name: drop-down list. The provider parameters are declared in 'lookup.xml'. Type a new name to overwrite.
- Class: type of the parameter (java class name).
- Default value: the value to be passed to the event provider at startup.



Generic parameters (real-time providers):

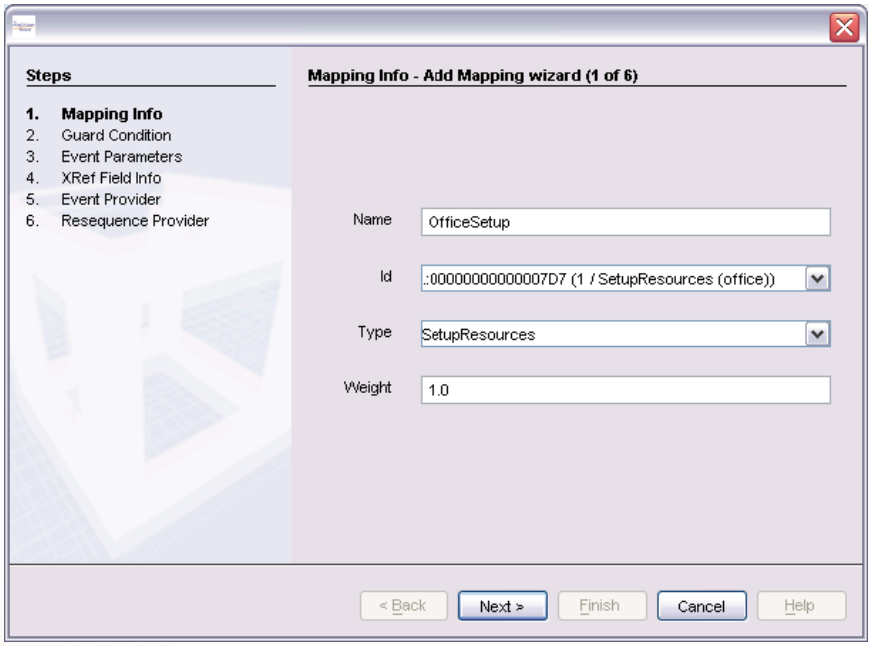
Property	Description	Default
topic	Topic on which the resequence engine provider will send events.	Mandatory
minimumInterval	Minimum time interval between the resequence event. In order to assure the correct order of the events, the resequence.	1000

Add mapping wizard

'Add Mapping' wizard allows new event mappings to be added to the mapping configuration.

The wizard takes the user through the steps described below.
First step, 'Mapping Info' panel requires input for the following fields:

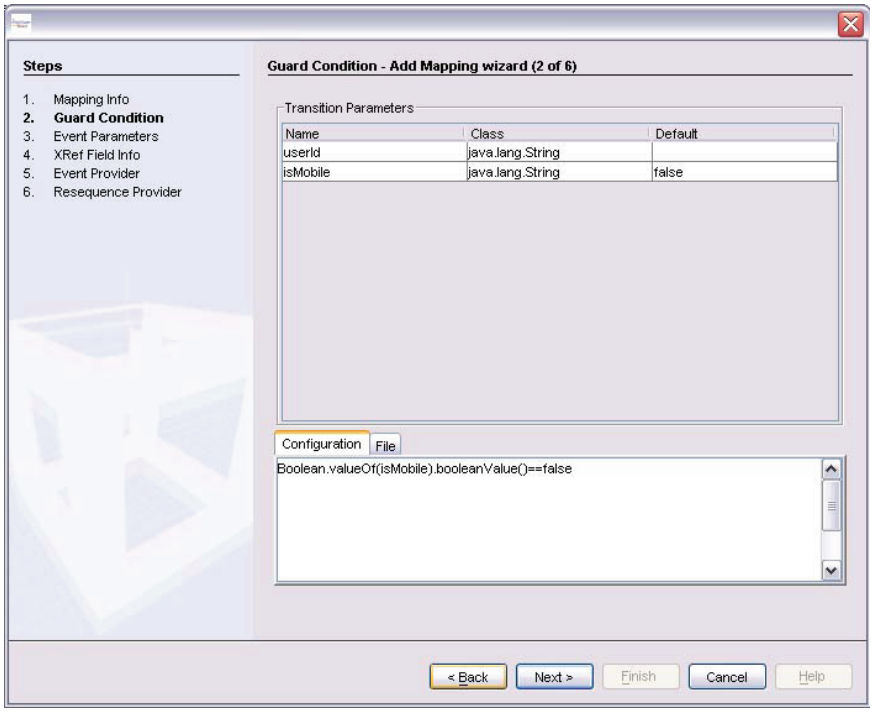
- Name: the declared name of this event mapping.
- Id: select from a list of event id's as declared in XMI. The drop down list will mark with a green background color all the transitions that have a guard condition associated. To overwrite, type an ID. After an ID selection, the type field will be updated with a matching type for the selected transition.
- Type: select from a list of declared event types. To overwrite, type an event type as a java class name.
- Weight: Double between 0 and 1 representing the probability of the event. 1 is the highest.



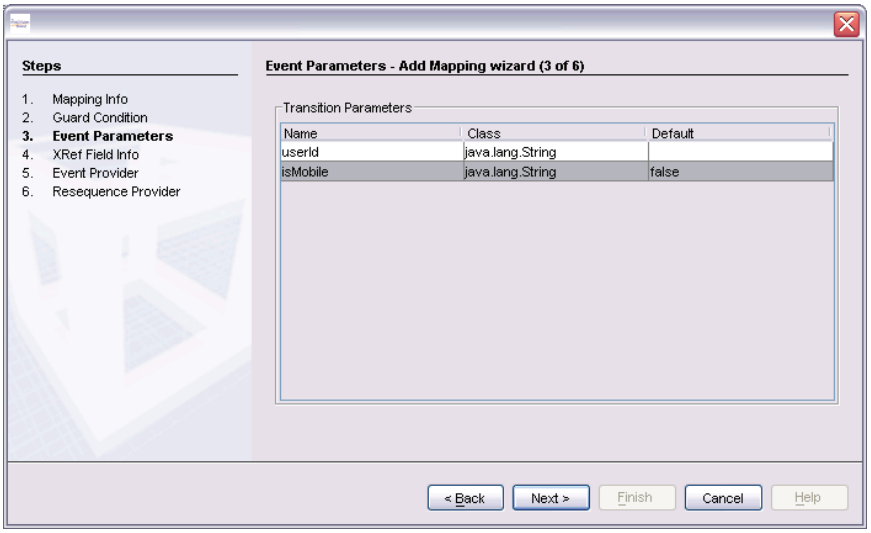
Second step, 'Guard Condition' panel allows editing of the guard transition parameters.

Guard transition parameters are initially populated with the guard condition parameters defined in the process definition file.

Panel is identical to 'Guard Condition' panel from 'Edit Mapping' section of this documentation.

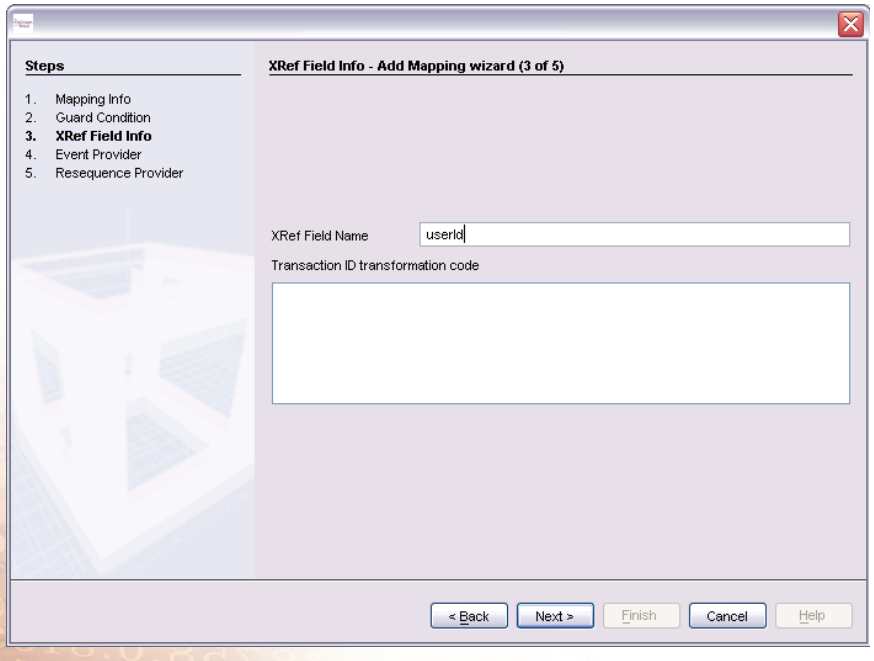


Third step, 'Event Parameters' panel allows editing of the calculated event parameters.

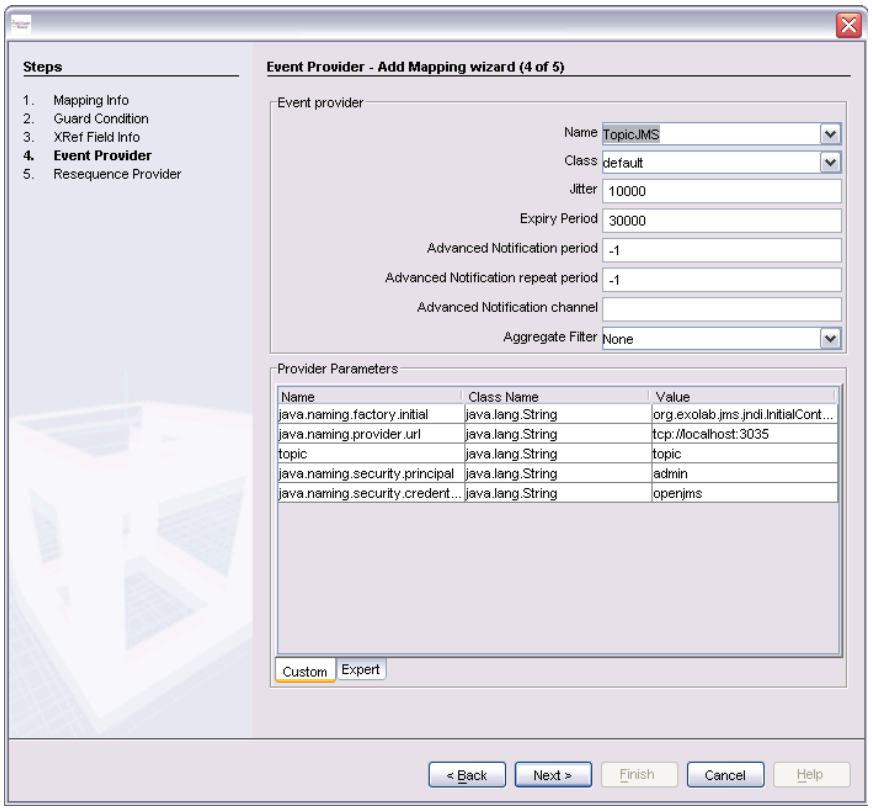


Next step, 'XRef Field Info' panel requires information about xRef field, that represents one of the event fields used to uniquely identify a transaction.

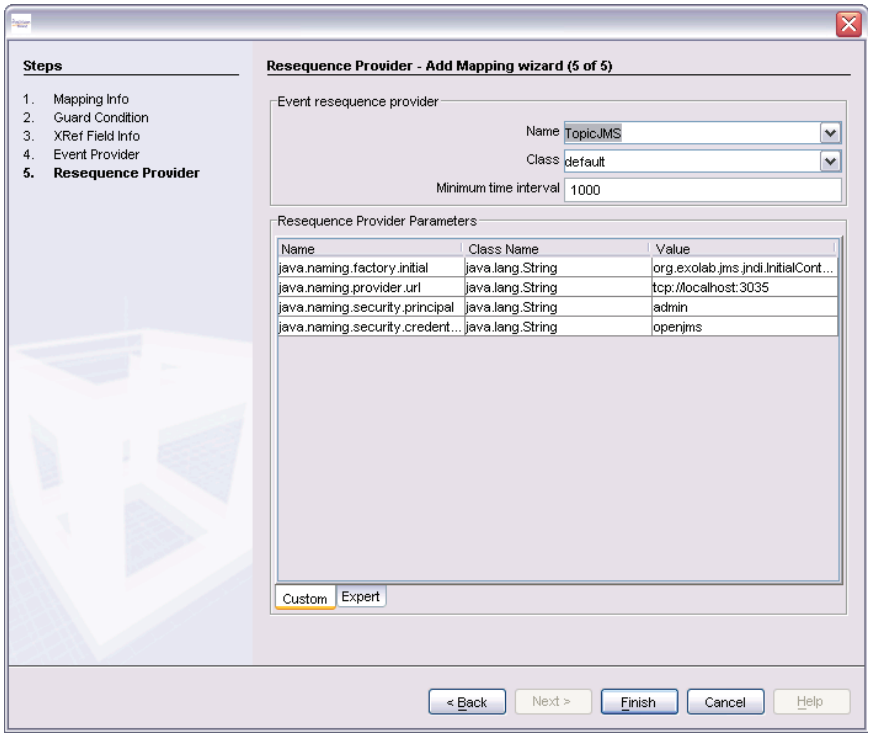
If the value of the xRef field represents the result of a computation, enter the transformation code inside 'Transaction ID transformation code' text area.



'Event Provider' panel creates an event provider and defines its parameters. This panel is identical to 'Event Provider' panel described in the 'Edit Mapping' section.



Last step, 'Resequence Provider' panel defines the resequence provider and its parameters, if needed.



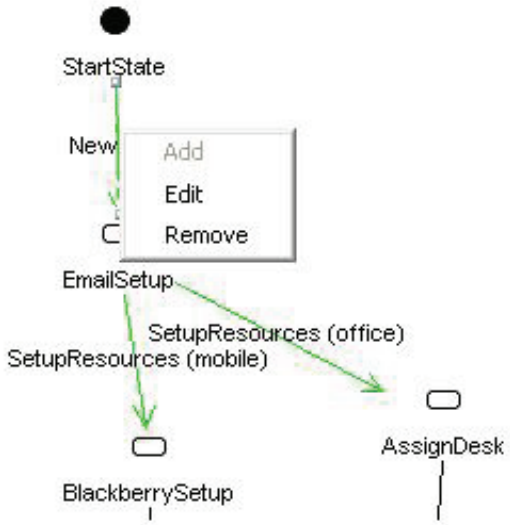
Process Graph

In the 'Process Graph' panel, the graph represents the structure of the process.

Transactions that have a guard condition specified are marked in green colour.

Graph transactions have a context menu associated. Menu options are the same as the ones from the mapping table's context menu ('Generic Mapping' section), as follows:

- Add – if a corresponding mapping already exists in the mapping table, this option is invalidated. Otherwise, an 'Add Mapping' window will open.
- Edit - an 'Edit Mapping' window will open, that allows the user to make changes to the selected mapping.
- Remove – removes corresponding mapping from the mapping table.



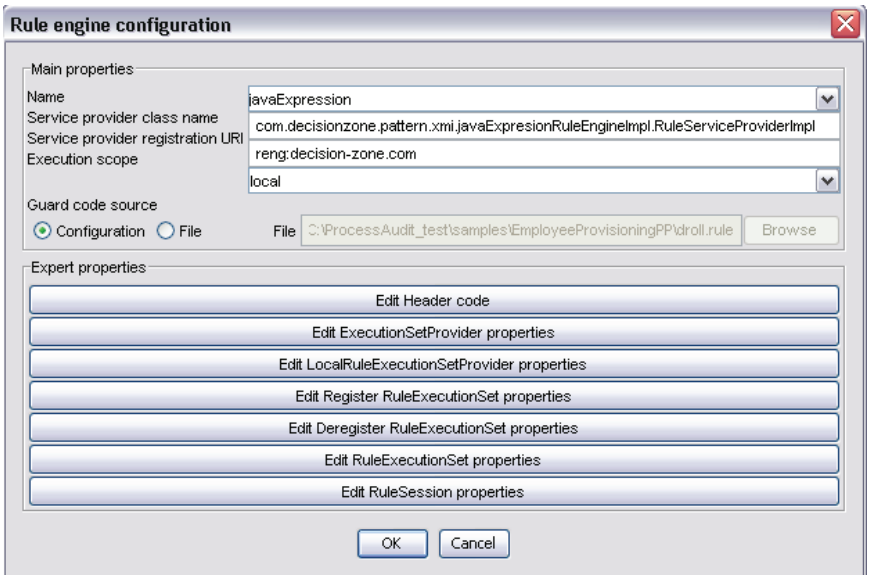
Selection of a transaction inside the graph will automatically select the corresponding row in the mapping table, if there is any. Also, any selection of a row from the mapping table will determine selection of the corresponding transaction inside the graph.

Rule Engine configuration

The DZAudit engine is designed to use a standard java rule engine (JSR 94). Out of the box the engine comes with two preconfigured engines:

- "javaExpression" engine, a custom made rule engine that executes simple java expressions as rules (default rule engine).
- "drolls", jBoss rule engine.

To configure the rule engine, select 'Edit' button from the 'Rule Engine' panel.



'Main properties' panel groups together the most common properties of the rule engine:

- Name: a drop down field containing all the available rule engines.
- Service provider class name: auto populated field that contains the rule engine provider implementation class name. User can modify this field.

Rule Engine configuration

- Service provider registration URI: auto populated field containing the registration URI. User can modify this field.
- Execution scope: drop down field with the following possible values:

o local, the scope of guard condition code is reduced only to the current transition. Only fields from the current transition event trigger are available on guard condition code evaluation.
o global, the scope of guard condition code is extended to the whole transaction. All the fields from all preceding transition trigger events are available to the guard condition code evaluation.

- Guard code source:

o configuration, the guard code is taken from the mapping configuration.
o file, guard codes for all the events are grouped together in the specified file.

'Expert properties' panel groups together buttons that allow access to all the properties defined by JSR 94.

"javaExpression" Rule code description

If the default rule engine "javaExpression" is used with the 'Configuration' option selected, based on guard code snippets the engine will generate a rule code that is stored in COMPILE_TEMP folder of the DZAudit Server.

Generated code sample:

```
/*automatically generated code. do not modify*/
import java.lang.String;
import java.lang.Integer;
import java.lang.Boolean;
import java.lang.Long;
import java.util.*;
import com.decisionzone.businessbroker.event.IBPEvent;
public class C_EmployeeProvisioning{
```

```
private Hashtable eventsTable = new Hashtable();
public void addEvent(IBPEvent event) {
eventsTable.put(event.getProperty("eventName"), event);
}

public void removeEvent(IBPEvent event) {
eventsTable.remove(event.getProperty("eventName"));
}

public boolean T_SetupResources__office____0000000000000830(
java.lang.String userId,
java.lang.String isMobile) throws Exception {
IBPEvent currentEvent = (IBPEvent)eventsTable.
get("SetupResourcesForOffice");
if ((Boolean.valueOf(isMobile).booleanValue()==false)){
return true;
}else{
return false;
}
}

public boolean T_SetupResources__mobile____0000000000000833(
java.lang.String userId,
java.lang.String isMobile) throws Exception {
IBPEvent currentEvent = (IBPEvent)eventsTable.
get("SetupResourcesForMobile");
if ((Boolean.valueOf(isMobile).booleanValue()==true)){
return true;
}else{
return false;
}
}
}
```

If 'File' option is selected, the user has to provide a code for the guard conditions evaluation. The generated sample code could be used as a template for the guard condition code and the following rules should be respected:

- the name of the class must be C_<name of the process definition>
- all the imports must be present in the new code.
- eventsTable, addEvent, removeEvent must be present in the new code.

o eventsTable is a Hashtable object containing the current event, plus all the events that caused the current event if the execution scope is set to global. The Hashtable's keys are represented by String objects having the event mapping name as value. The Hashtable's values are represented by an IBPEvent object, which is a bean like representation of the event.
o addEvent, removeEvent are convenience methods used by the rule engine to add and remove events from the events table.

- Transition guard rule method name must be T_<normalized name of the transition>. The normalized name of the transition is (name + "_" + id).replaceAll("\\W","_")
- Transition guard rule method must declare as parameters all the event fields defined as guard parameters in the event mapping.
- Transition guard method result must be boolean.

"drools" Rule code description

A sample drool rule code:

```
package test
import com.decisionzone.businessbroker.event.IBPEvent;
import com.decisionzone.pattern.xmi.engine.GuardConditionStatus;

rule "T_SetupResources__office____0000000000000830"
when
event : EmployeeProvisioning_SetupResources(isMobile == "false")
```

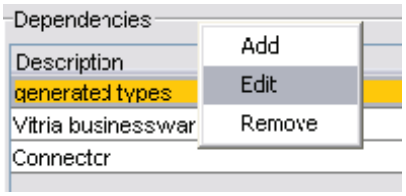
```
guardConditionStatus : GuardConditionStatus(eventName ==
"SetupResourcesForOffice")
then
guardConditionStatus.setStatus(true);
end

rule "T_SetupResources__mobile____0000000000000833"
when
event : EmployeeProvisioning_SetupResources(isMobile == "true")
guardConditionStatus : GuardConditionStatus(eventName ==
"SetupResourcesForMobile")
then
guardConditionStatus.setStatus(true);
end
```

Drool code rules:

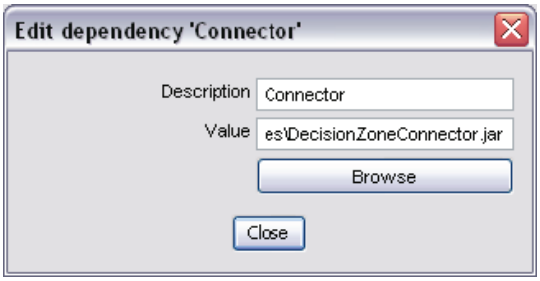
- all the imports must be present in the new code.
- Transition guard rule method name must be T_<normalized name of the transition>. The normalized name of the transition is (name + "_" + id).replaceAll("\\W","_")
- The name of the events is the name of the associated event mapping.
- For each event the rule engine automatically associates a 'guard ConditionStatus' object. The 'guardConditionStatus' has a String field named 'eventName' that represents the corresponding event and a Boolean field 'status' that represents the guard condition status.

Dependencies

Dependencies are resources required at runtime phase (jars/classes/…). To edit them, click 'Edit Dependencies' button that will open a new window. Select the required action from the menu associated with right click on dependencies table/rows.

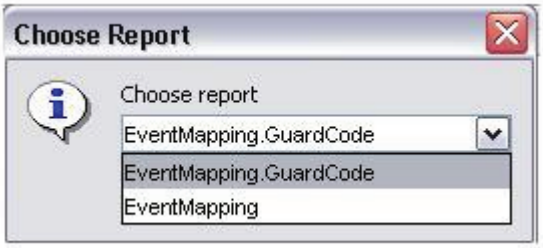


Edit screen:



Report

To view a HTML report using the default browser on your machine, select 'Report' option from the 'Event mapping' node's context menu, then select the desired report from the list.



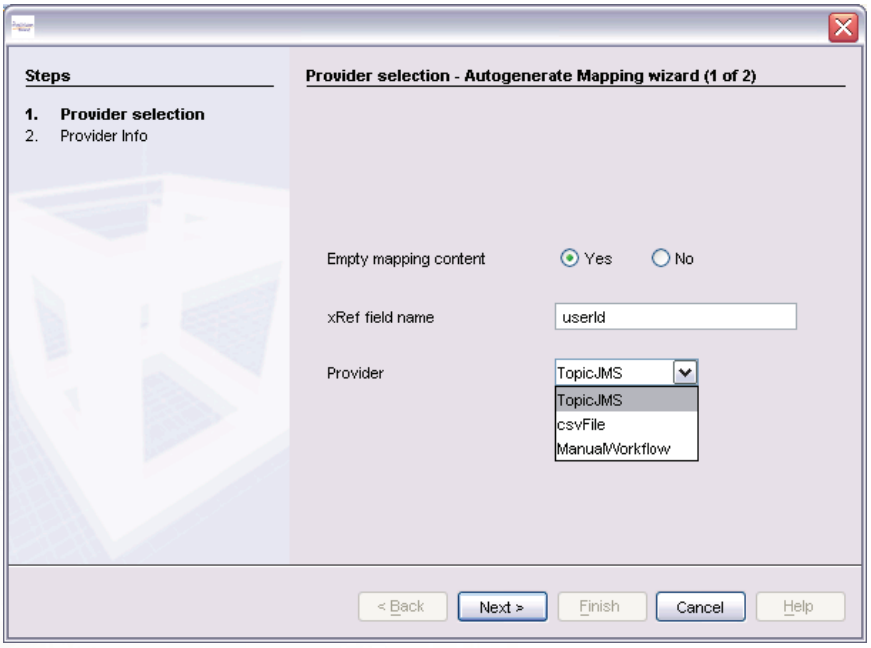
Autogenerate Mapping

This option is a wizard that will automatically populate the event mapping file (.pam) based on the process definition. Each transition defined in the process will become an event in the mapping file, and the parameters of the transition will translate into event parameters.

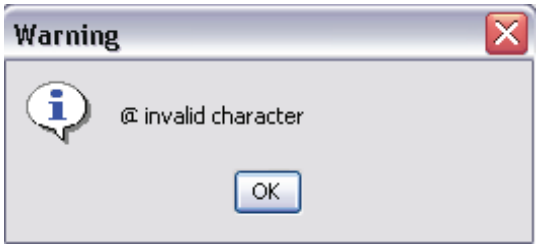
The wizard's initialization window will accept a manual entry for the xRef field name, a selection of the provider to be used and a radio selection that will determine if the existing content of the mapping file to be deleted or not.

The xRef field name and the provider selected will be saved for each event entry created in the mapping file.

If user chooses not to empty the existing mapping content, all the event entries generated will be appended to the existing ones.

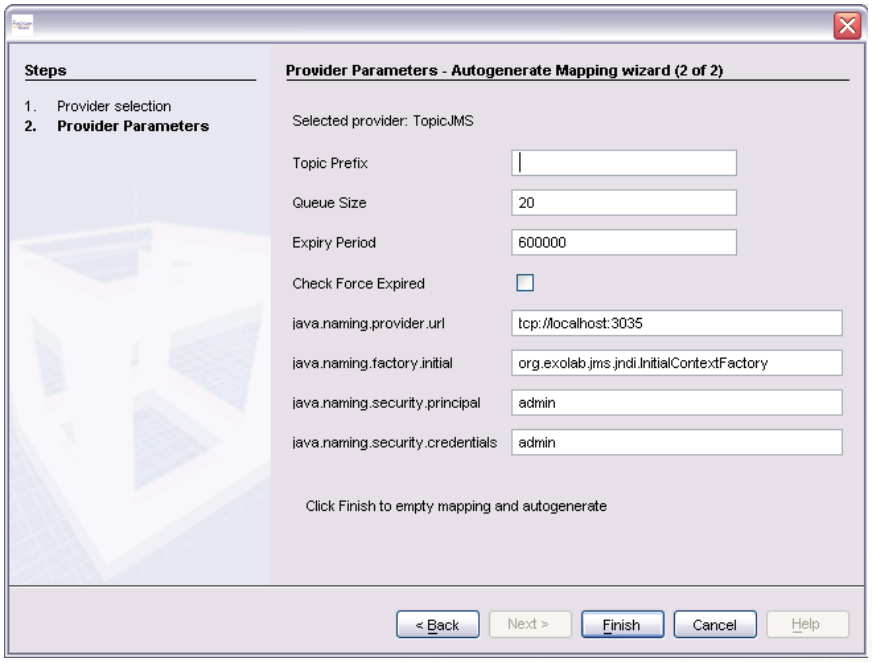


If an invalid character is entered for xRef field name, a warning message will be displayed.

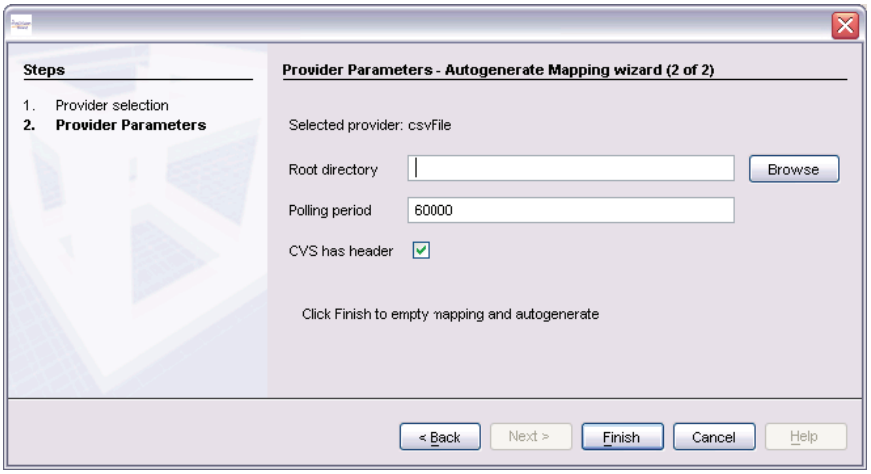


Next window will display parameters specific to the selected provider, as shown below. Some of the parameters have default values; others await the user's input.

'TopicJMS' provider parameters:

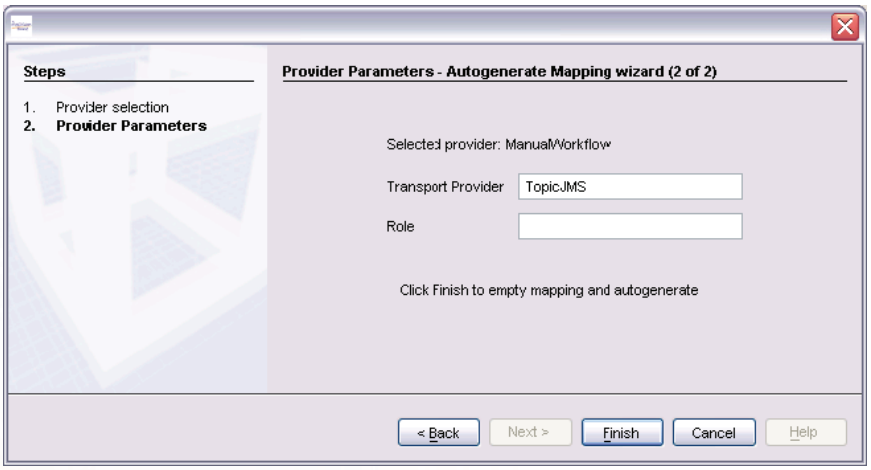


'cvsFile' provider parameters:

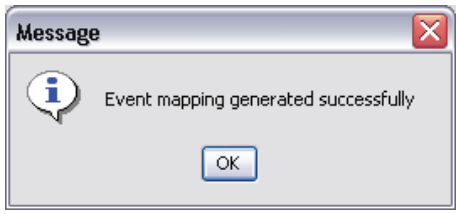


'Browse' button will open a file explorer window displaying the system's directory structure, so the user can pick the directory of interest.

'ManualWorkflow' provider parameters:



Click 'Finish' to start the autogeneration process. If the mapping file is successfully generated, a popup message will be displayed. If an error occurs, an error message will be displayed.



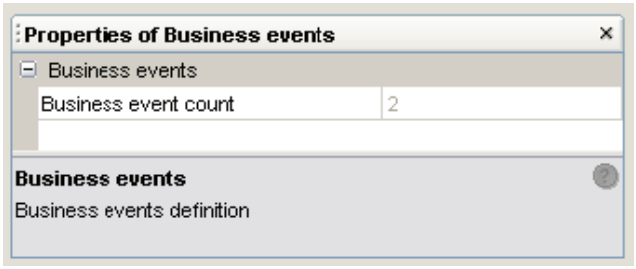
Note: *For more information regarding the selected provider's parameters and their description, please visit chapter 'Event Providers'.*

Descriptor – Business events

After a valid process definition XMI file and an event mapping file were specified, the descriptor is ready to be deployed in a runtime environment (server).

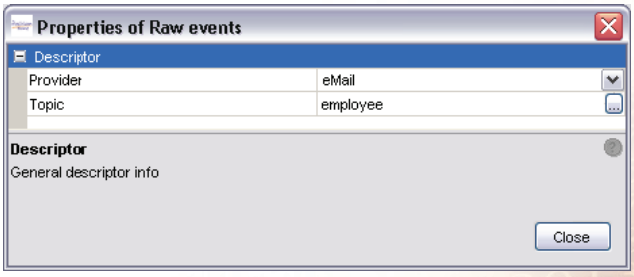
The result of the correlations and pattern matching in a form of event sets will be published on differed middleware layers. This section controls the way these sets are published and when.

Properties displayed if the Business events node is selected.



Raw events

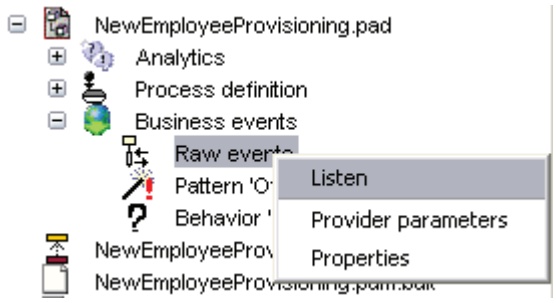
For 'Business events' section of a descriptor there is always a child node called 'Raw events' that controls the publishing of the event sets resulted from the correlation process (normal or anomalies).



Properties:

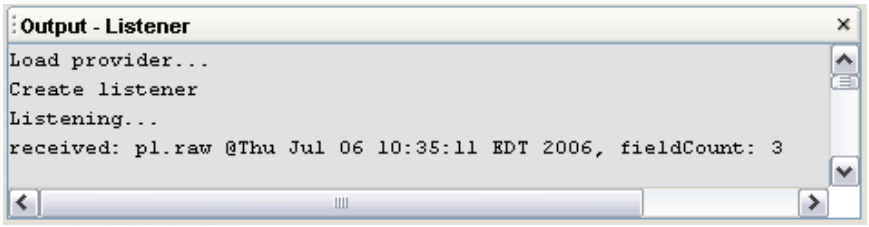
- Provider: provider name as declared in 'lookup.xml' to be used to publish the correlated event sets.
- Topic: the topic to publish the messages to. Must be compliant with the conventions of the selected event (middleware) provider.

To facilitate debugging and understanding of the correlation process, by selecting 'Listen' from the node context menu, an event set inspection panel will be displayed.

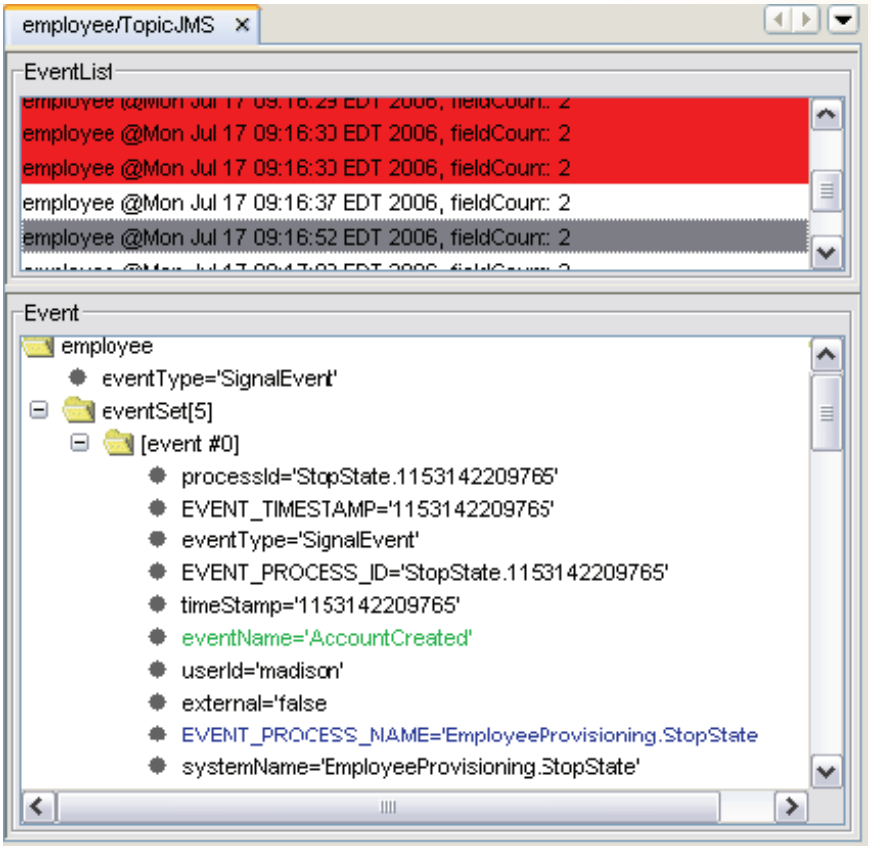


The listening process can also be started from the 'Tools' menu, by selecting 'Listen' option (see Chapter 5 'Tools', under 'Listen option').

The output window will display the status of the listening process.



The inspection panel will display on the upper side the list of time-stamped events and the number of events in its set, and by selecting the required row, a detailed view can be accessed in the lower side of the panel.



Any event set that contains a Boolean field called 'anomaly' set to 'true' value, will be highlighted with a red background color.

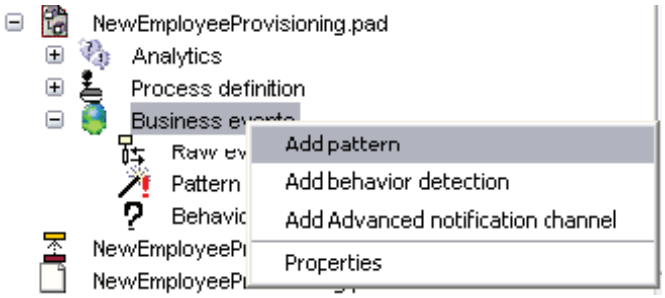
The 'eventName' fields are highlighted with green fonts.

Blue font color highlights the added correlation fields.
The listening process can be stopped by closing the active window or by selecting 'Stop listening' from the right click menu that appears when clicking on the event list window. The listening process can be restarted by selecting 'Start listening' from the menu.

Patterns

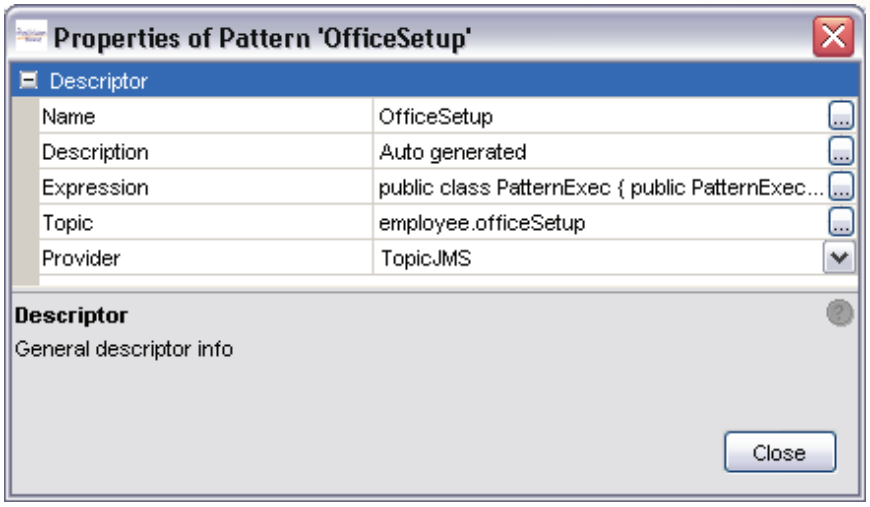
After an initial correlation was performed, further filtering can be performed by applying pattern expressions. The result will be expressed as subsets of events from the original raw event set.

To create a pattern business event, the user can follow the analytics process or select 'Add pattern' from the context menu associated with 'Business events' node.

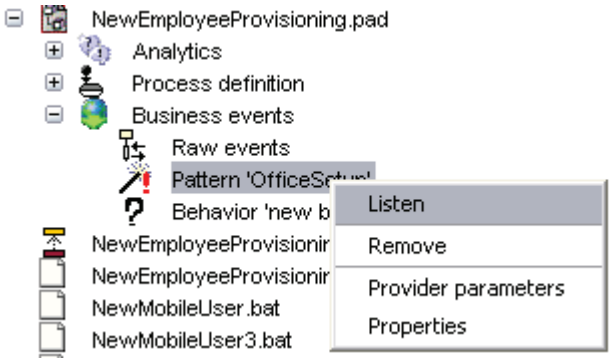


Properties:

Property	Description	Default
Name	Pattern name (business event name)	NA
Description	Short description of the pattern expression	Auto generated (if generated by analytics process)
Expression	'Rapide' compliant pattern expression. Please consult 'Rapide' user manual for details on the 'Rapide' language syntax.	
Topic	Topic on which the eventual pattern match (event set) will be published.	NA
Provider	Event provider name as declared in 'lookup.xml'	



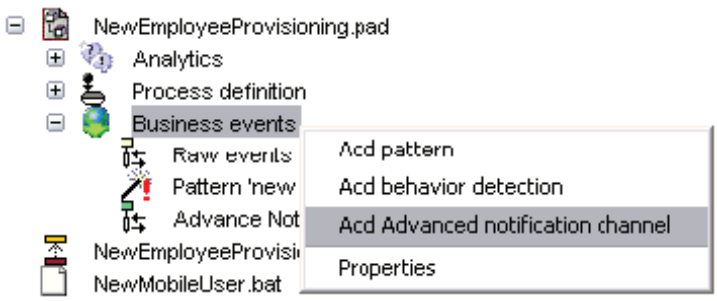
To facilitate debugging and understanding of the correlation process, by selecting 'Listen' from the node context menu, an event set inspection panel will be displayed.



The usage of the message listening panel is identical to the listening panel for 'Raw events'.

Advanced Notifications

To create a new advanced notification channel, the user must select 'Add Advanced notification channel' from the context menu associated with 'Business events' node.



Properties:

Property	Description	Default
Name	Advanced notification channel name.	
Description	Short description.	
Topic	Topic on which the advanced notifications will be published.	
Provider	Event provider name as declared in 'lookup.xml'	

The usage of the message listening panel is identical to the listening panel for 'Raw events'.

Descriptor – Analytics

Analytics functionality is used primarily to identify events of interest and assist in pattern creation (Business event).

Note: *This section is optional and has no impact on the process audit runtime.*

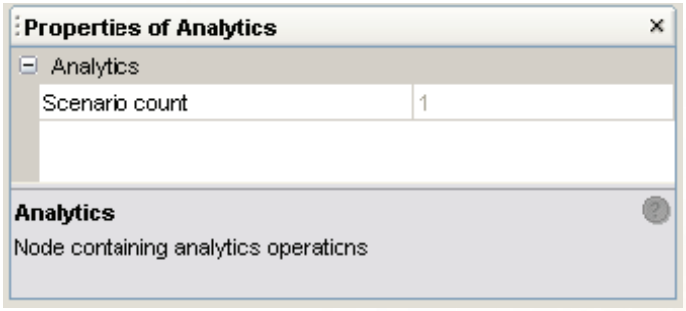
Scenario

A scenario contains ONE capture file reference, a filtering section (Event list) and multiple model nodes.

Once a capture process is completed, out of the resulting data, during the exploration phase, multiple models can be generated. These models will be used to identify the events of interest in the larger set of events in the capture file.

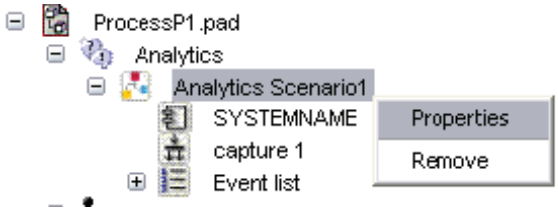
To add a new scenario, select 'Add scenario' from the context menu associated with the 'Analytics' node:

The properties of an 'Analytics' node show the number of defined scenarios.



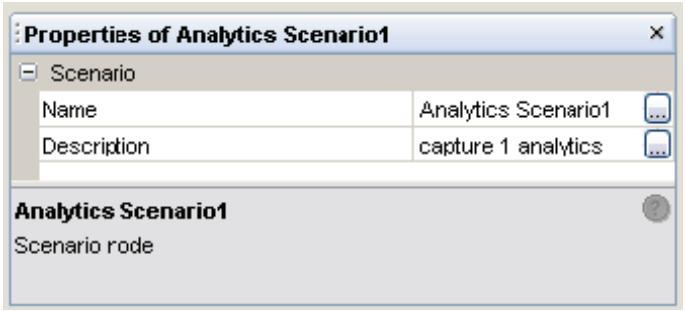
Scenario *(cont'd from p 53)*

To remove a scenario, select 'Remove' from the associated context menu:



Properties:

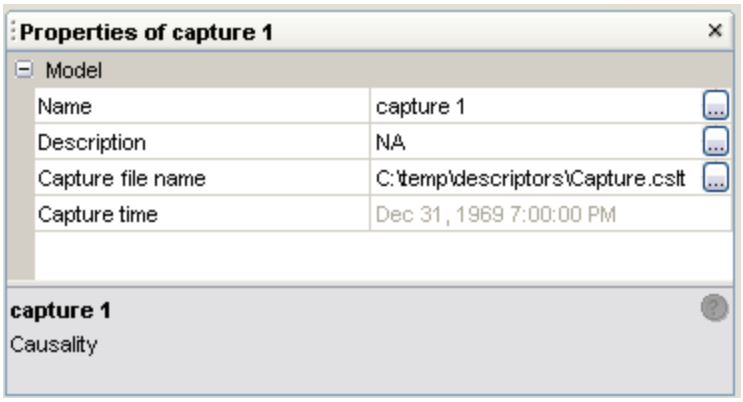
Property	Description	Default
Name	Scenario name	
Description	Short description of the scenario	



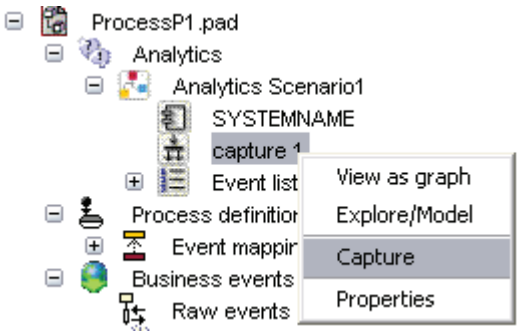
Causality – Capture

A capture file can be specified by setting the 'Capture file name' from capture node properties.

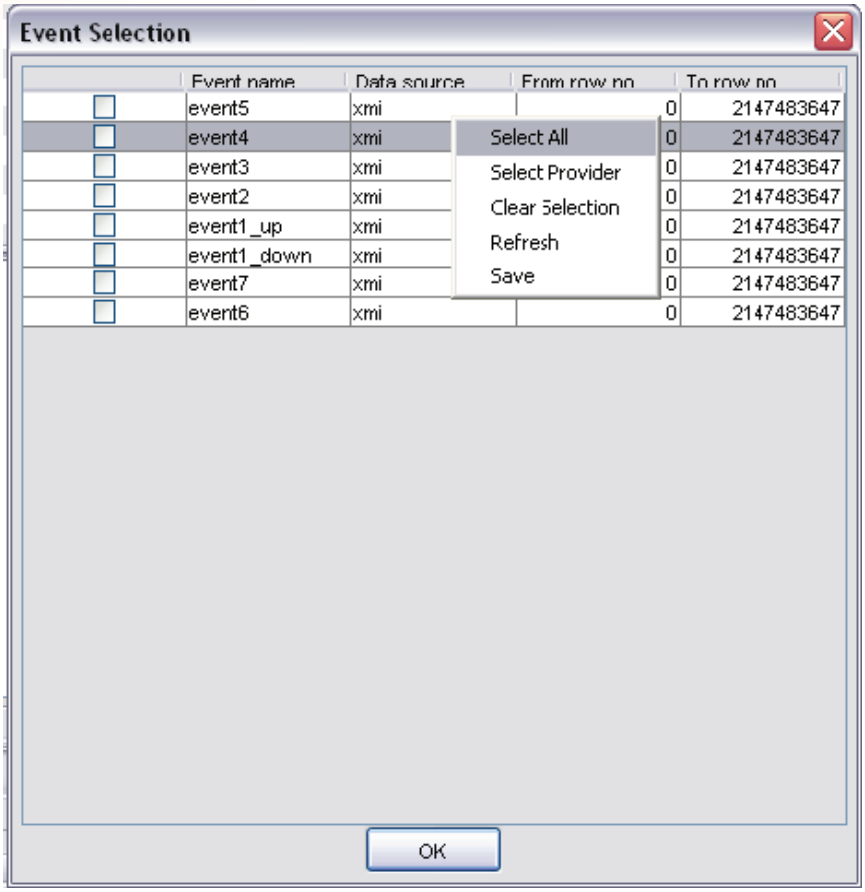
If none is available, a new one can be created using 'Create new resource' procedure.



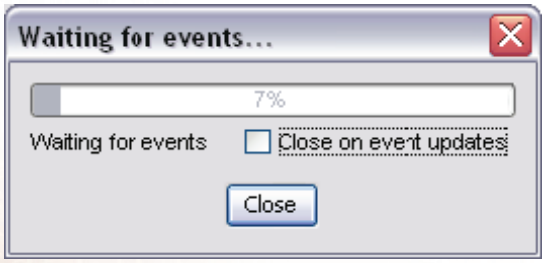
To capture a dataset to the associated file, select 'Capture' from the associated context menu.



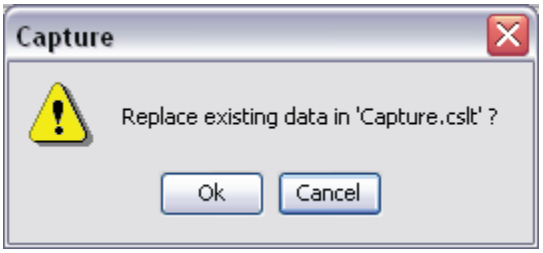
Select the event of interest from the presented list:



To capture multiple events ensure that the 'Close on event updates' is unchecked. When satisfied with the number of events captured, click 'Close' to continue. If 'Close on event updates' is selected, the progress window will close automatically after the first event set is received.

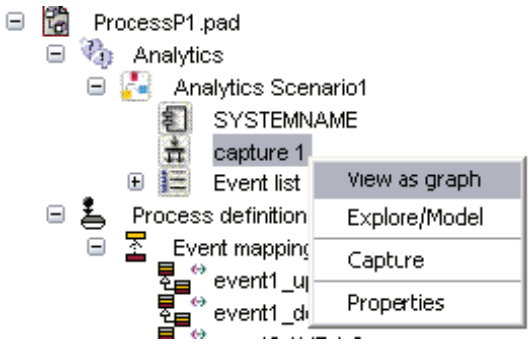


Confirm or cancel the procedure:



Causality – Explore

Data from a captured dataset (causality data) can be explored in a multidimensional way by selecting 'Explore/Model' from the context menu of a causality data node.

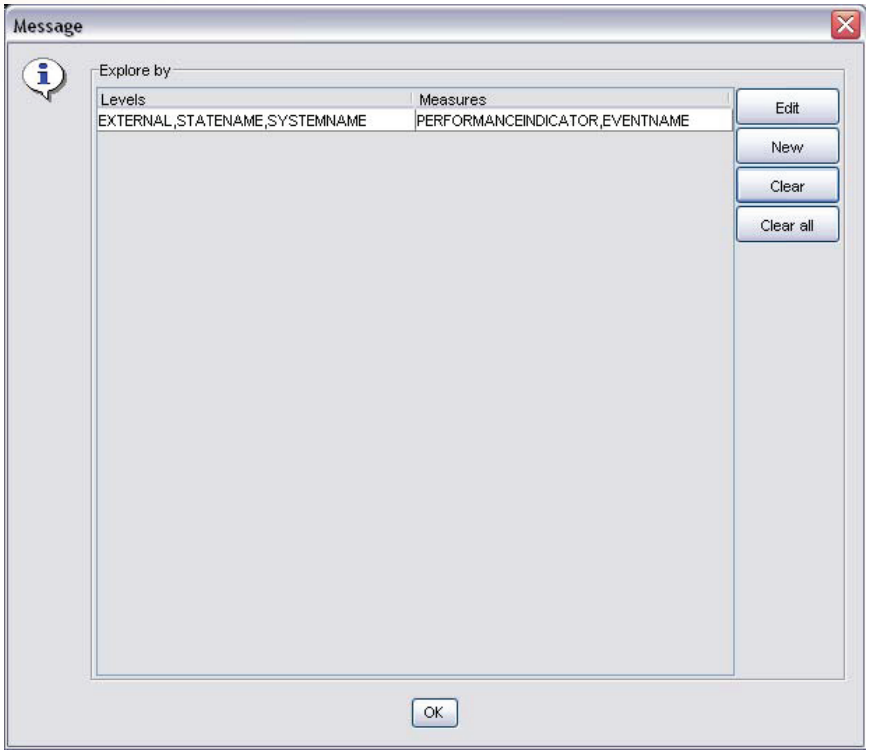


The exploration is done using a synchronized tree-table view.

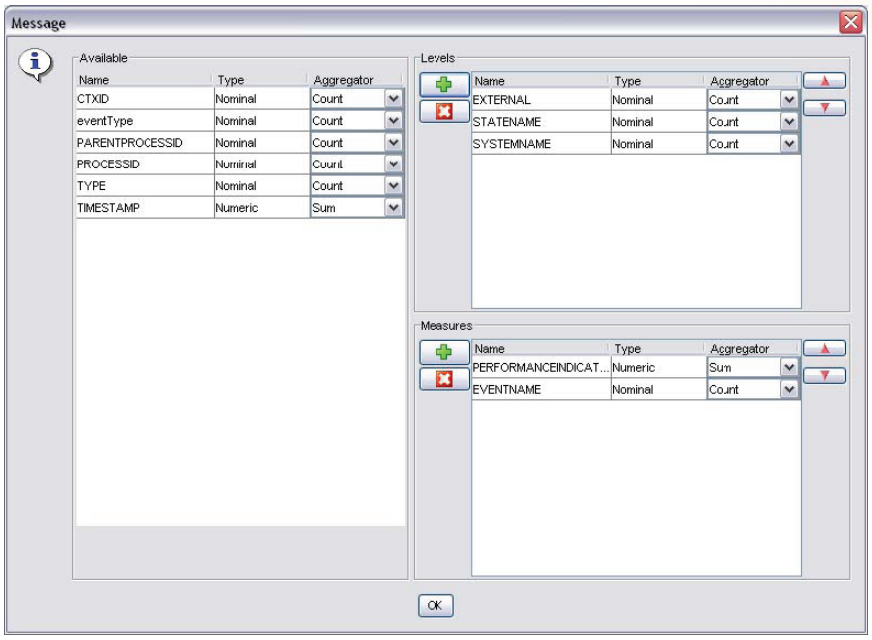
The sequence of drill down categories/fields is called a hierarchy.

The next screen allows the user to select one of previously used hierarchies or to add, edit or remove an existing one.

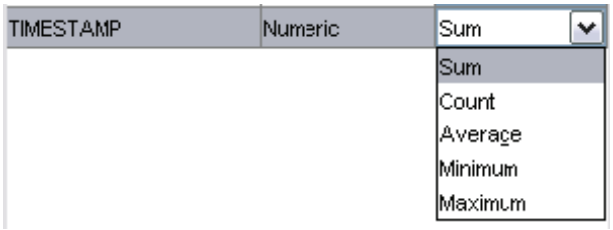
All hierarchies defined during a session will be cleared after IDE shutdown.



If a new hierarchy is to be created, the configuration screen displays all available fields in the dataset. These fields can be added to levels or measures category. If a field will be part of the levels, then that field will be included in the tree part of the view, otherwise it will be listed in the table part of the view.

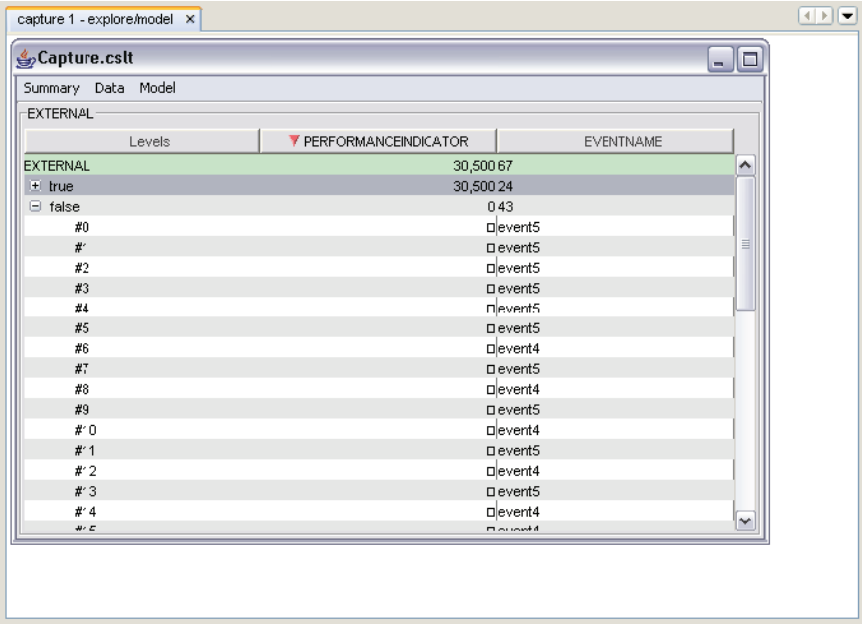


While a section of the tree-table view is collapsed, the value of all its associated measures (hidden) will be rolled-up (aggregated) using a formula specified by a drop-down selection.



Note: *Nominal values can only use 'Count' algorithm.*

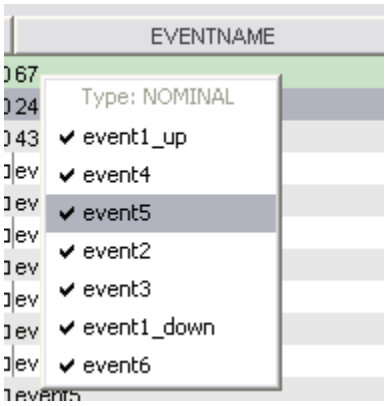
Figure showing a sample view.



The sorting order of the measures can be changed by clicking on the associated table header. The red triangle icon will show the sorting column and the direction of the sort. Clicking repeatedly on the same column header will toggle between ascending and descending sorting.

The type of the measure can be displayed if the user right-clicks on the column header.

If the measure is of nominal type, then the shown menu will allow the user to filter the dataset by the selected nominal values. All deselected values will be excluded from calculations and display.



To show a histogram view or a XY plot of the field being explored, select the required option (view) from the 'Summary' menu:

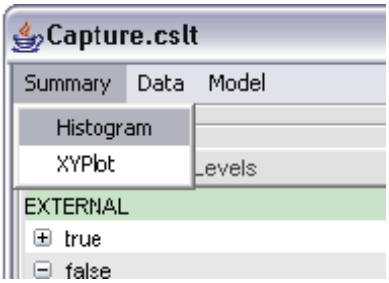


Figure showing the histogram of all the fields:

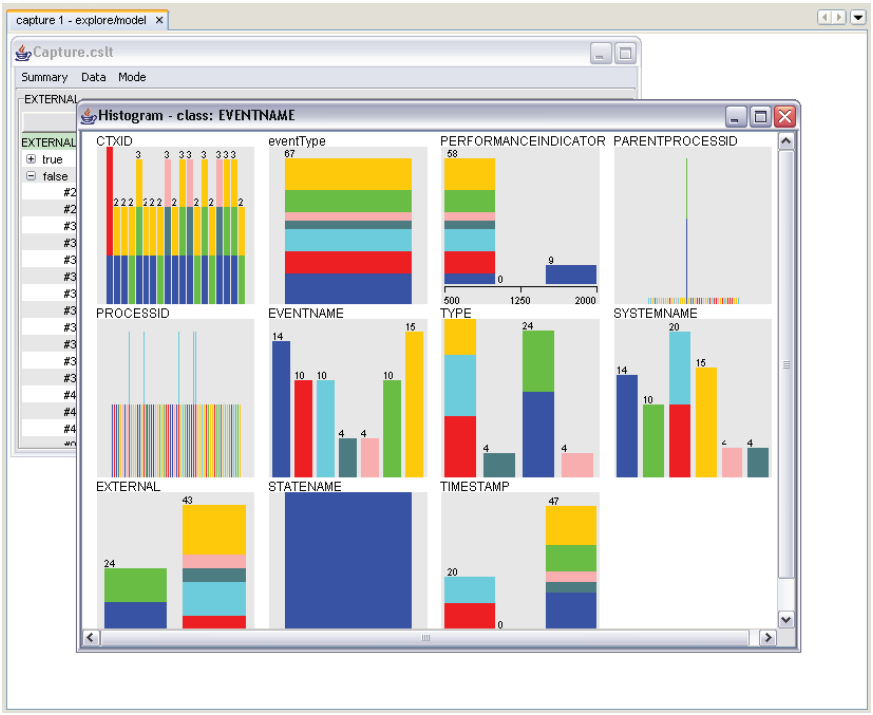
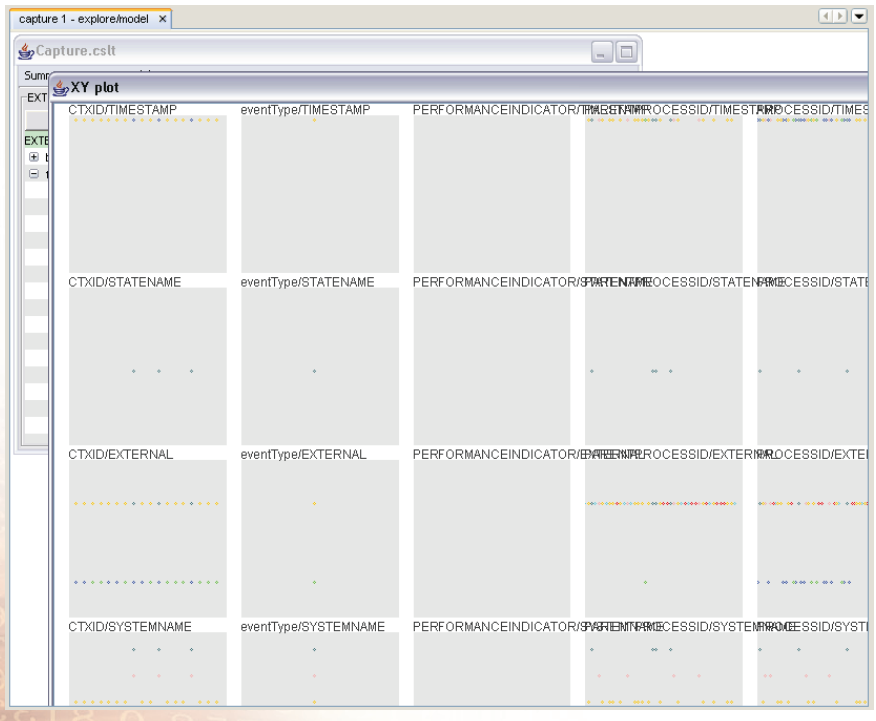


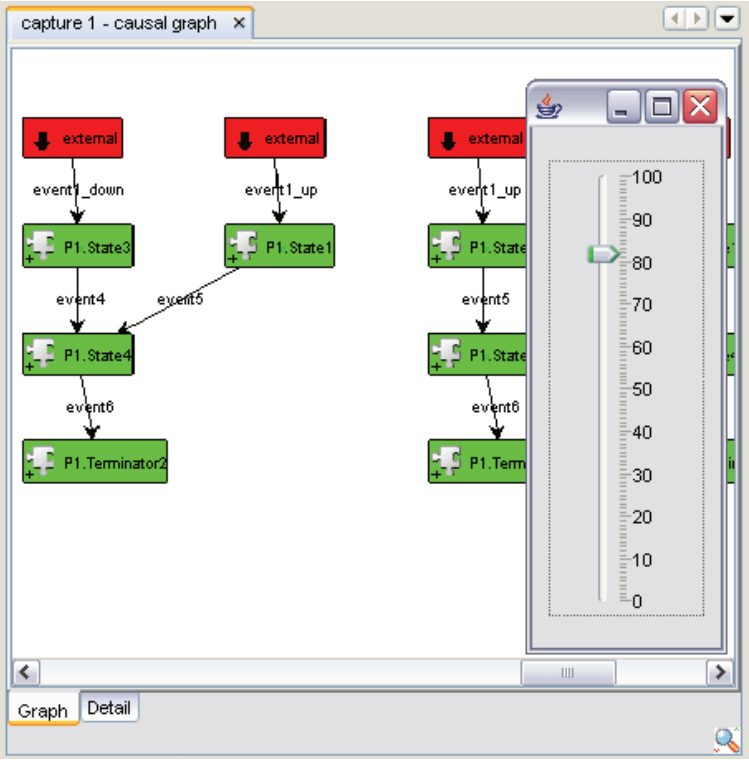
Figure showing the XY plot of all the fields:



Causality – View as graph

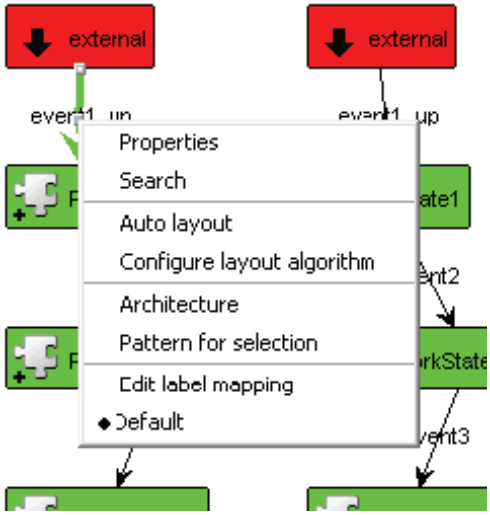
Data from a captured dataset (causality data) can be explored as a causal graph by selecting 'View as graph' from the context menu of a causality data node.

The red states represent the external states out of which the event has triggered the state change. The transitions represent the events that triggered the state change.

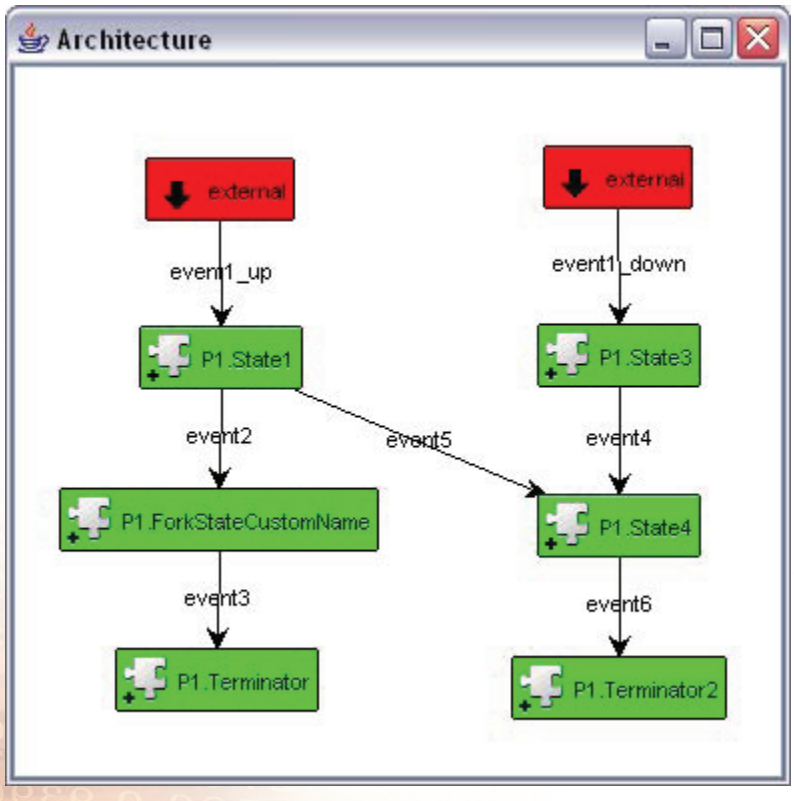


Note: *The small magnifying glass controls the zoom level. 'Auto layout' will trigger the automatic arrange of the elements on the screen.*

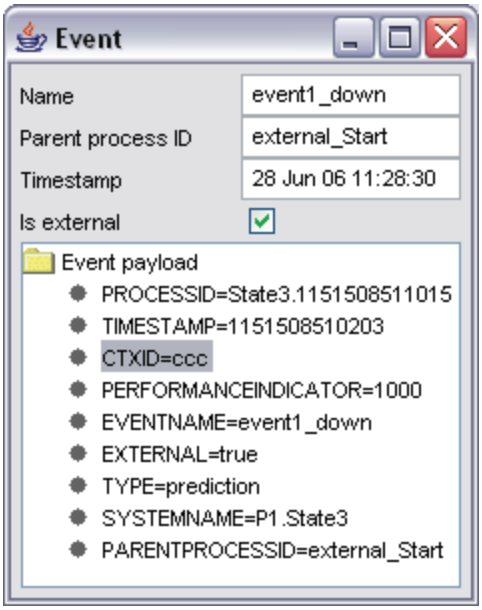
To bring up the context menu, right click on an event or empty space.



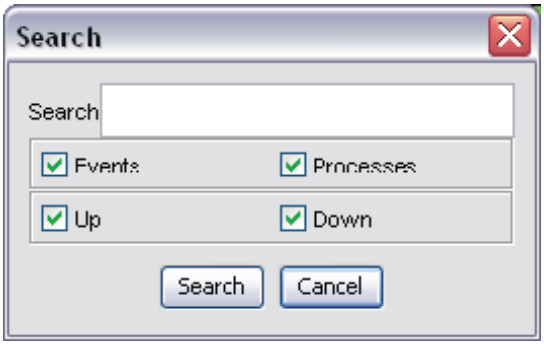
Architectural view triggered by 'Architecture' menu selection shows the unique states and the detected transitions between them:



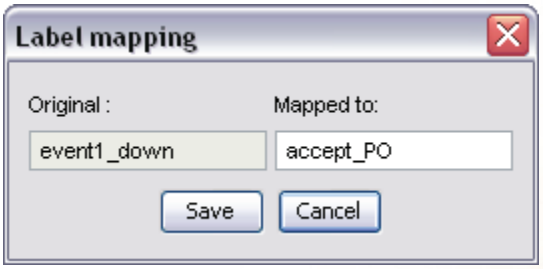
Event properties view triggered by 'Properties' menu selection if the right click was performed on a transition:



Find event ('Search' selection from context menu):



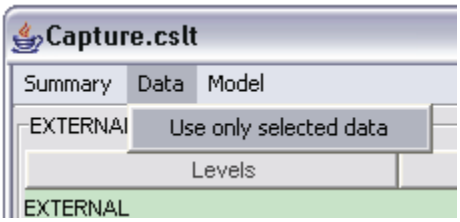
Edit event label mapping option from context menu will allow the user to specify business level labels for any of the captured events:



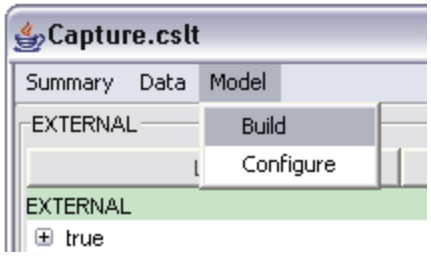
Causality - Model

During the exploration of the data from a captured dataset using the multidimensional view, multiple models can be generated (built) for the selected measures. The currently selected measure (sorted column) will be considered as the output class of the dataset. The model will be built for that specific measure.

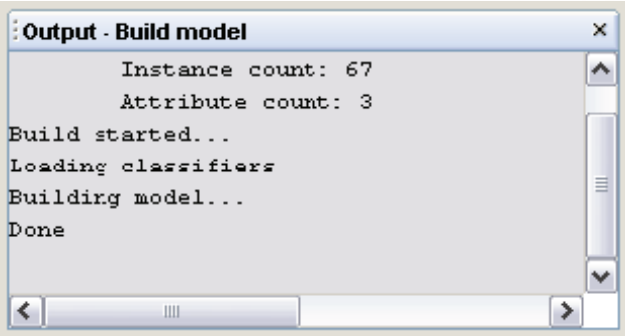
The tree-table view supports multi interval selections. If the option 'Data'->'Use only selected data' is enabled, then the model will be built only from the data included in the selections.



To trigger the model creation, select 'Model' -> 'Build' from the frame menu.

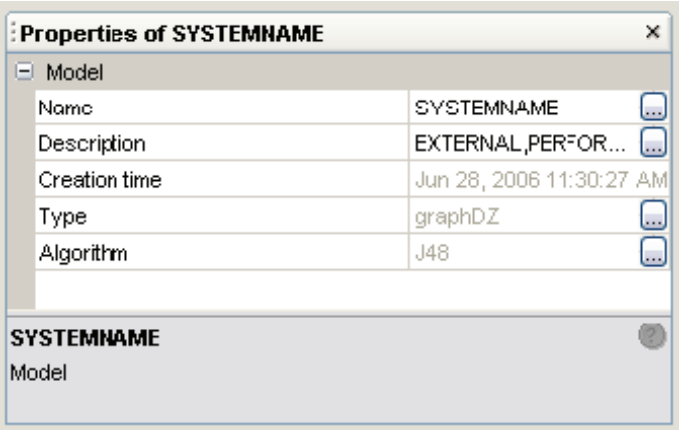


The output window will show the status messages associated with the model build. Upon completion, the resulted model will be stored in a sibling node of the causality capture file, with the name of the selected output class. The text representation of the model will be shown in a window beside the data explorer.



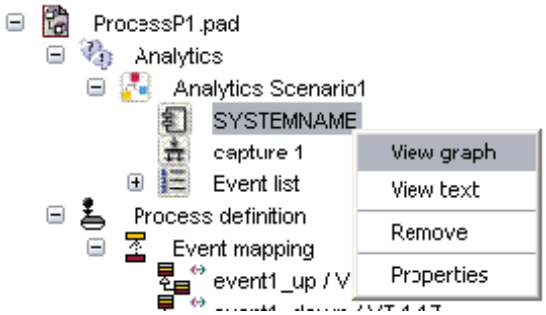
Properties of the generated model:

Property	Description	Default
Name	Name of the model	Selected measure
Description	Model sort description	Hierarchy used while generating the model.
Creation time	Time stamp	Creation time
Type	Reserved	graphDZ
Algorithm	Reserved (type of algorithm used to generate the model – configurable)	M5P/j48

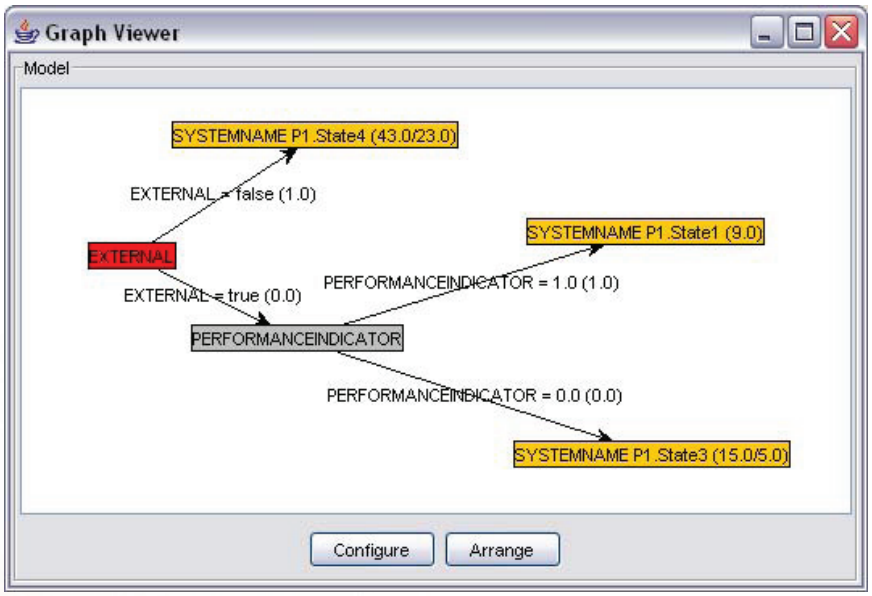


Model visualization

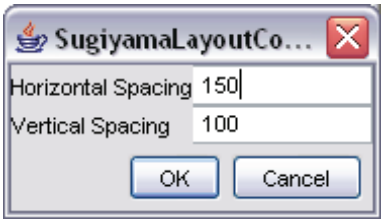
To visualize the structure of the model, select the required visualization style from the context menu associated with the model node



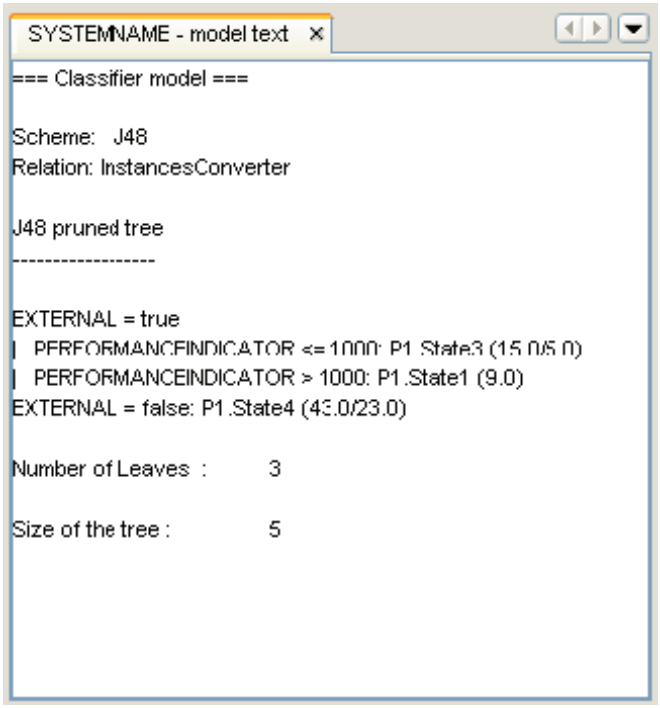
'View Graph' menu selection will popup a graph visualization window. Ex: decision tree model



Auto-arrange algorithm configuration:

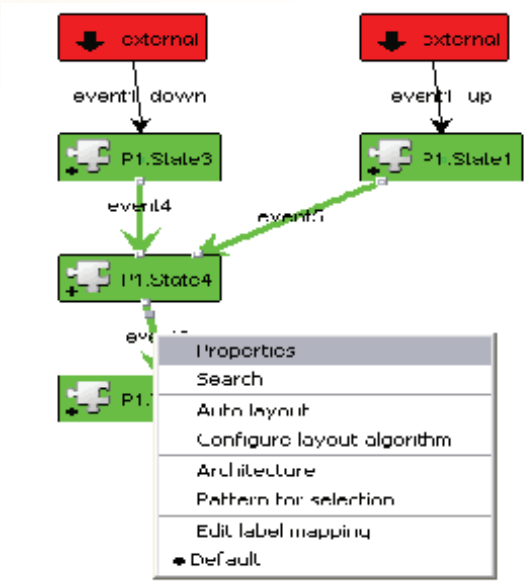


'View text' selection from the context menu will produce a textual representation of the model.

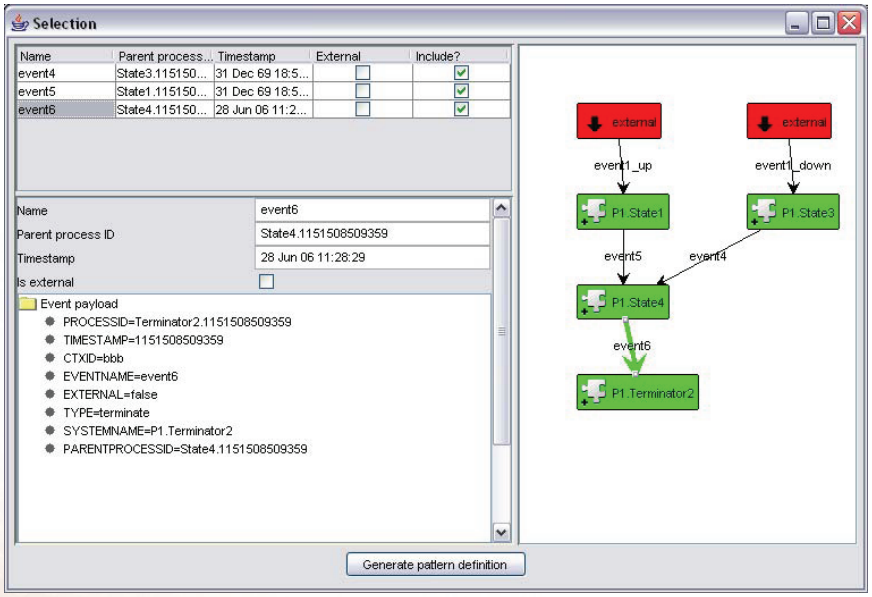


Pattern generation

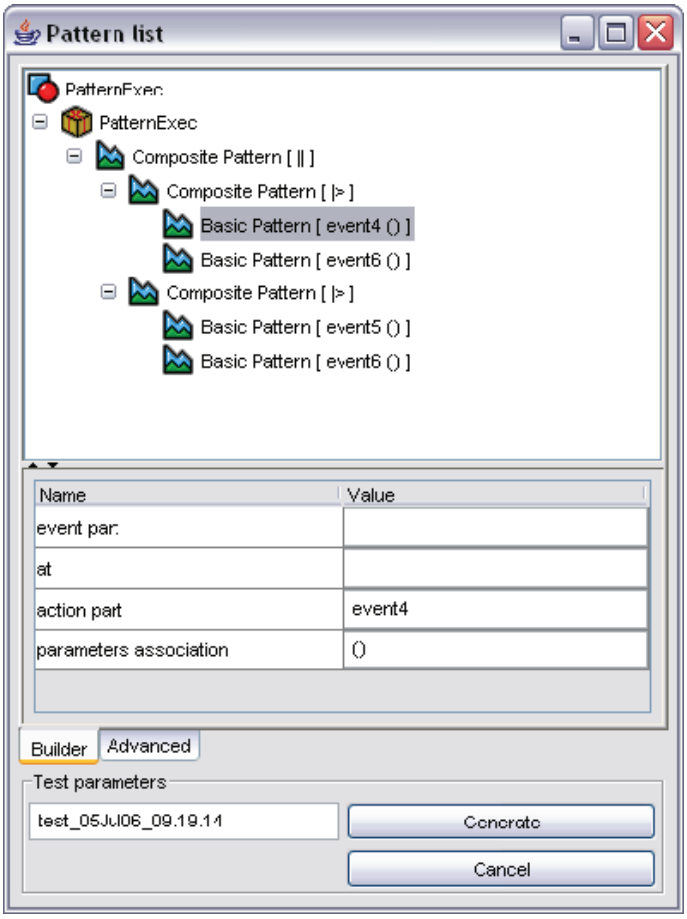
This section describes how the causal graph viewer can be used to generate the pattern expression skeleton.



While holding 'Ctrl' key down, select multiple events in the order of interest. The last click should be a right-click. This will bring the context menu. By selecting 'Pattern for selection' option, the event subset selection is displayed.



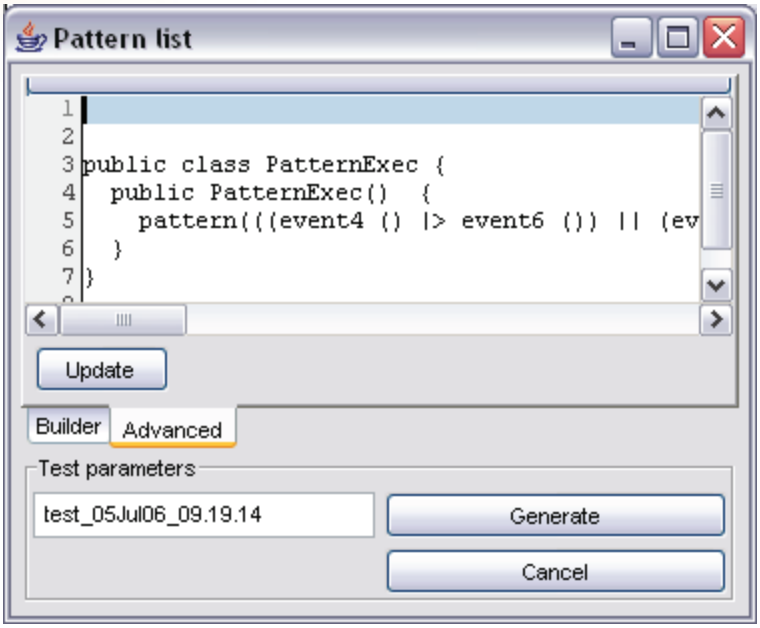
After reviewing and selecting all events of interest, the pattern skeleton structure will be displayed graphically:



The pattern expression can be edited by using the context menus and property fields. Please consult 'Rapide' user manual for details on the 'Rapide' language syntax.

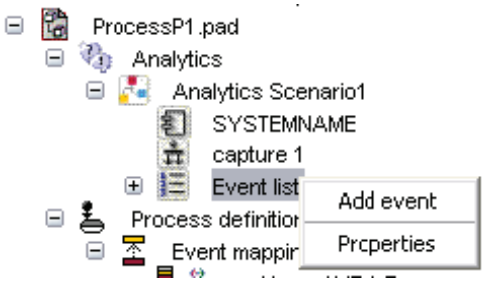
Change the name of the pattern under 'Test parameters' subsection. Click 'Generate' to create a pattern business event under the 'Business events' section of the descriptor node.

Alternatively, for advanced users, the view under 'Advanced' tab will reveal the pattern code expression. Full editing is now available.

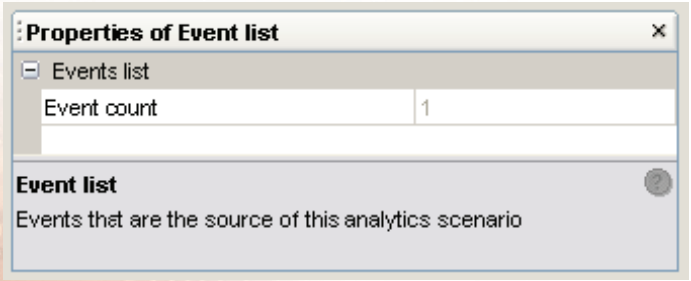


Event filtering

Reserved
Adding events



Event filter property.



Server configuration

To create a new configuration file for a 'Process audit server' follow the guidelines from 'Create new resource' section for a '.pas' file.

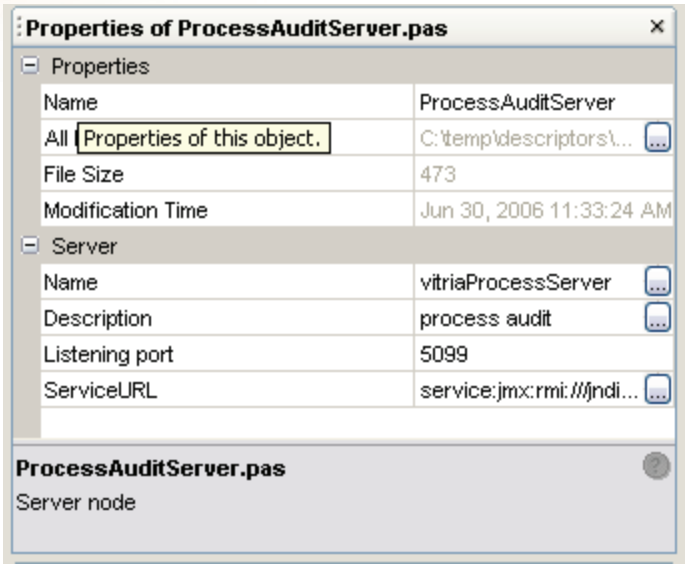
The server is a standalone process started by invoking the OS specific shell file. It is managed and queried using JMX.

Note: *Any JSR160 compatible JMX console can be used to query and manage the server remotely.*

Connection string: service:jmx:rmi:///jndi/rmi://[hostname]:[port]/ProcessAudit
Server MBean: com.decisionzone.processaudit:name=ProcessAudit Server
Descriptor MBean: com.decisionzone.processaudit.descriptor:name=[descriptor name]

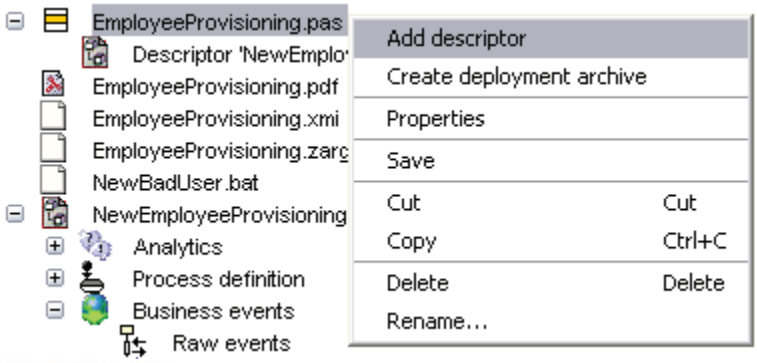
By selecting the server configuration '.pas' node, the properties panel allows the update of general parameters for the server.

Property	Description	Default
Name	Name of the server configuration	NA
Description	Short description	NA
Listening port	Port on which the server will listen for requests (JMX)	5099
Service URL	JSR 160 compliant connection string. Do not modify if uncertain.	



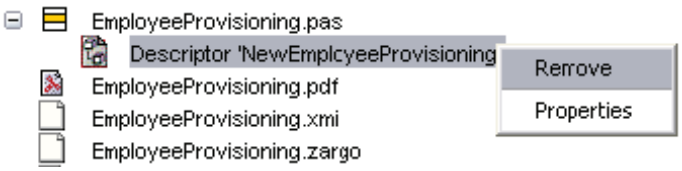
Add descriptor

To add a new descriptor to the list of the descriptors to be processed by this server, select 'Add descriptor' from the context menu:



Remove descriptor

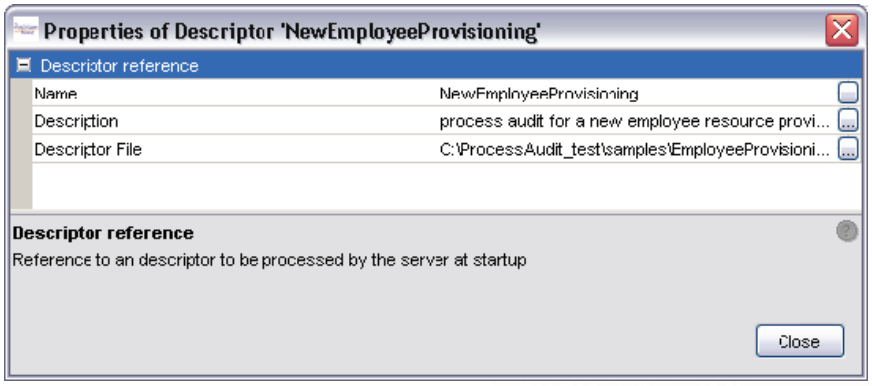
From the list of descriptors, right click on the descriptor to be removed, the select 'Remove':



Descriptor properties

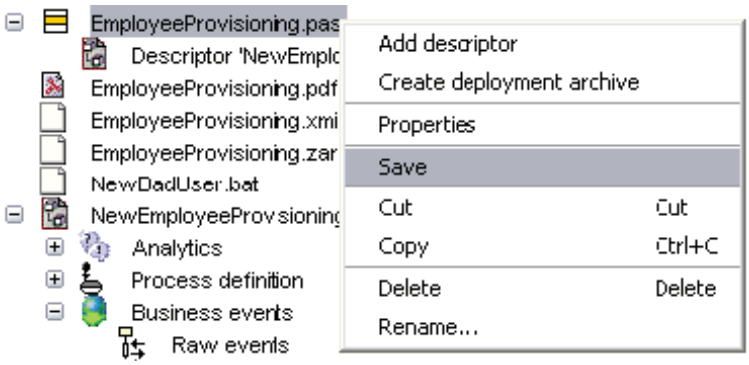
Properties of a descriptor entry in a server configuration file:

Property	Description	Default
Name	Name of the descriptor	
Description	Short description	
Descriptor file name	Reference to a descriptor file resource.	



Save server configuration

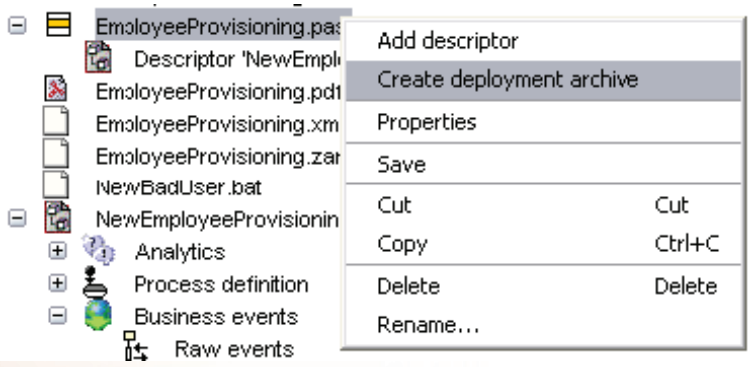
To save server configuration, select 'Save' from the context menu:



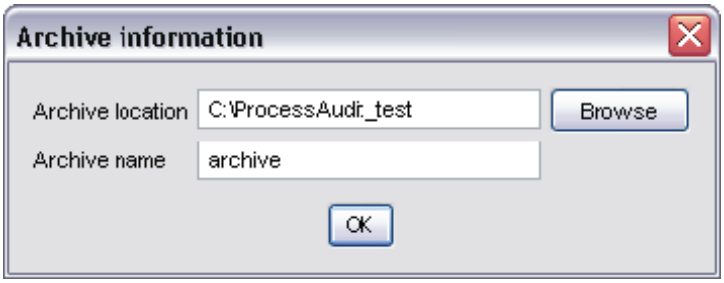
Create deployment archive

This option will create an archive ready for deployment, archive that contains the following files:

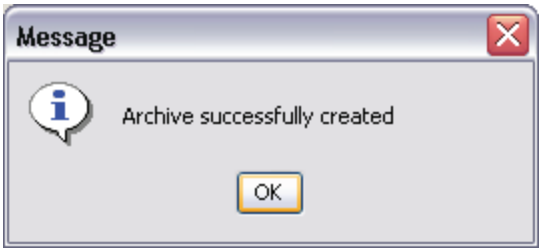
- one server configuration file (.pas).
- one or more deployment descriptor files (.pad), depending on the count of descriptor entries in the configuration file. Each descriptor file has a corresponding mapping file and process definition file.
- one or more mapping files (.pam).
- one or more process definition files (.xmi).



In the 'Archive Information' window, user has to input the archive's location, where the archive will be created, and its name, without any extension. A '.jar' extension will be automatically added to the archive's name.



If the archive is successfully created a message window will popup, otherwise an error message will be displayed.



Runtime

The runtime components for DZAudit are represented by at least one process audit server instance. This instance will load the descriptors specified in its configuration file at startup.

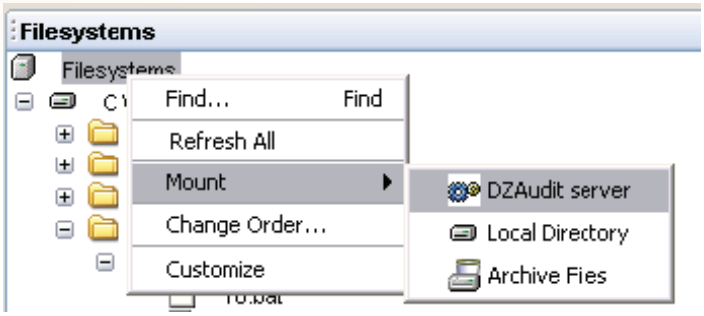
IMPORTANT: *If the number of descriptors declared in the server configuration file exceeds the maximum number of simultaneous process audits declared in the server license, the startup will terminate abruptly without running any descriptors. Contact your administrator or 'Decision-Zone' for additional information.*

Startup

Invoke the OS specific shell file with the name of the server configuration file ('.pas') as parameter. Make sure that the path environment variable contains the required entries for the specified descriptors and their resources.

Connect

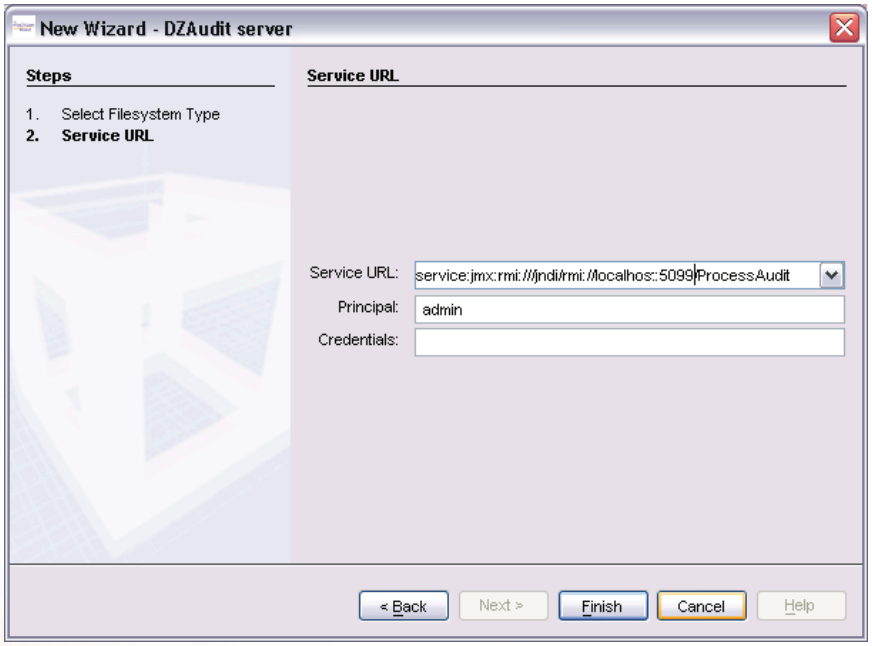
To connect to a specific server instance select 'File' -> 'Mount' or right click on 'Filesystems' node and select 'Mount' -> 'ProcessAudit server'.



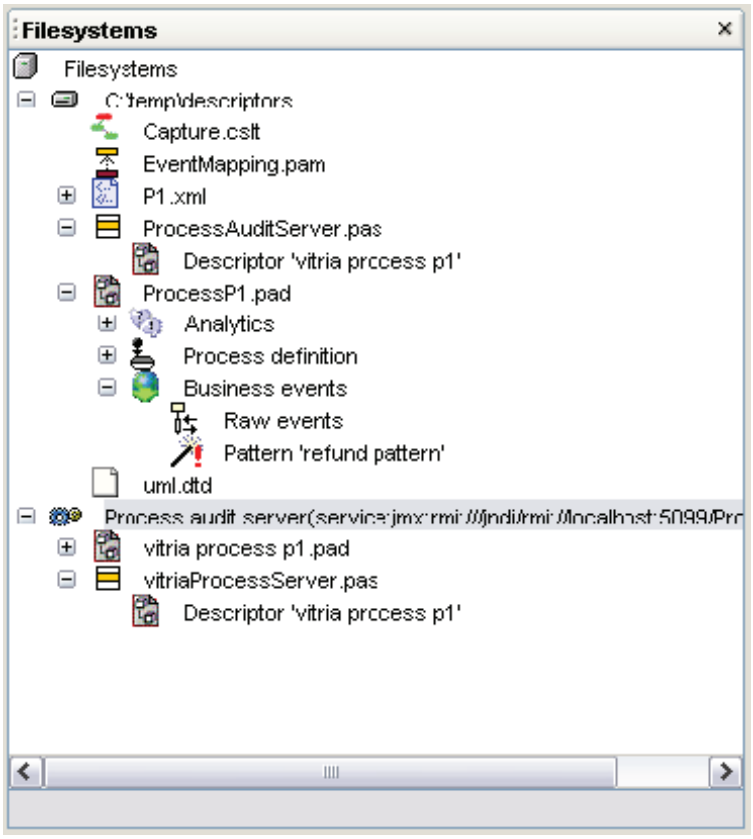
Type or select the service URL (connection string) in the drop down list. Usually only the hostname and port has to be specified for the service URL.

For JMX credentials, use username and password pair values specified in 'user.properties' file located in server's root directory. If the properties file is missing, no authentication is required.

Click 'Finish' to continue.

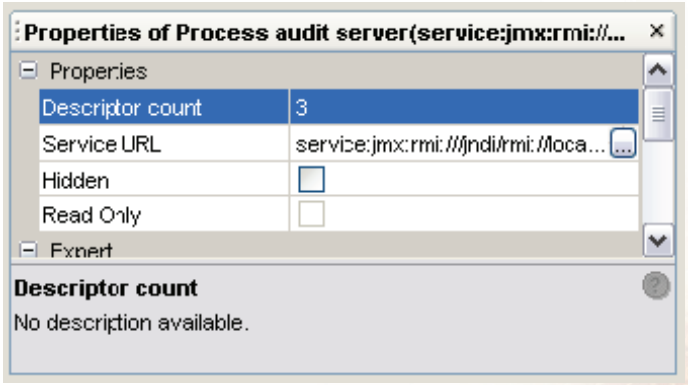


After the mounting process has been completed, the server should appear as a file system with the name 'Process audit server'.



By expanding the server node, all descriptors that were loaded at startup will appear as files.

Properties:



Management

The server is a standalone process started by invoking the OS specific shell file. It is managed and queried using JMX.

Figure showing the server MBean, its attributes and operations.

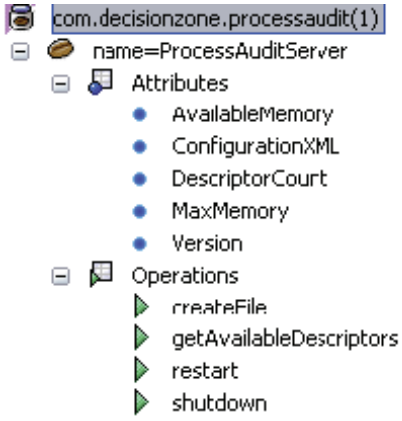
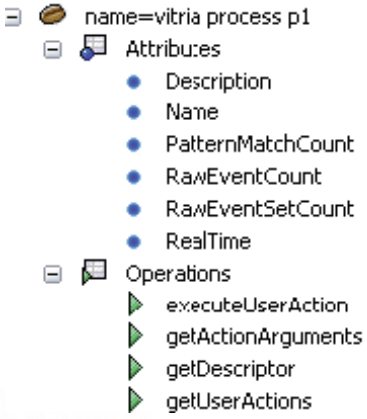


Figure showing a descriptor MBean, its attributes and operations.

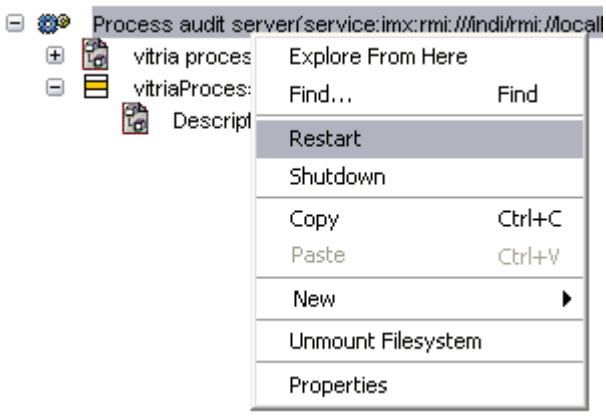


Note: *Any JSR160 compatible JMX console can be used to query and manage remotely the server.*

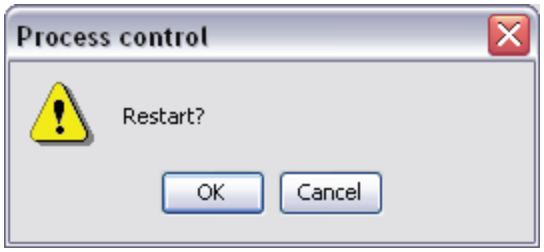
Connection string: service:jmx:rmi:///jndi/rmi://[hostname]:[port]/ProcessAudit Server MBean: com.decisionzone.processaudit:name=ProcessAuditServer Descriptor MBean: com.decisionzone.processaudit.descriptor:name= [descriptor name]

Restart

To remotely restart the server select 'Restart' from the context menu associated with the mounted server file system node.



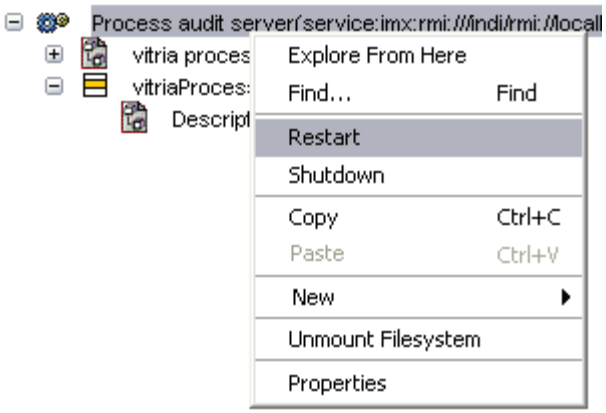
Confirm or cancel,



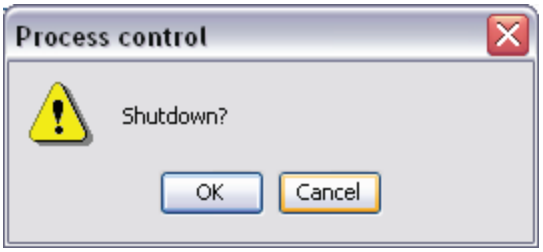
Status message of the restart command:

Shutdown

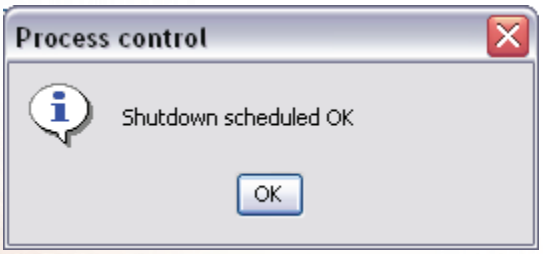
To remotely shutdown the server select 'Shutdown' from the context menu associated with the mounted server file system node.



Confirm or cancel,



Status message of the shutdown command:



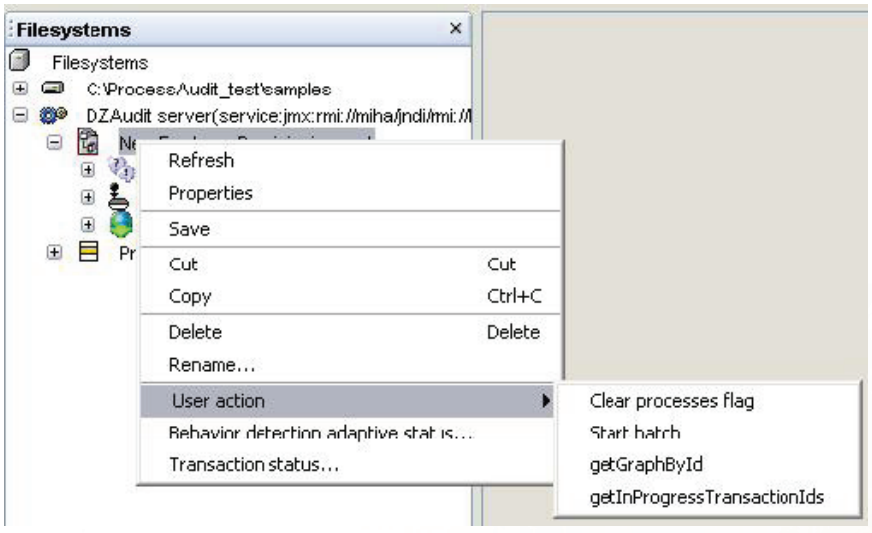
IMPORTANT: *After shutdown, the link will be severed abruptly. Unmount the server from the IDE (right click on the server file-system and select 'Unmount'). Default delay time is 20 seconds after the command has been scheduled.*

Executing user actions

Certain types of providers require the execution of 'user actions' remotely. As an example, the JDBC provider (exposes historical events stored in a JDBC compliant database as virtual events) requires the startup signal as a user action 'Start batch'.

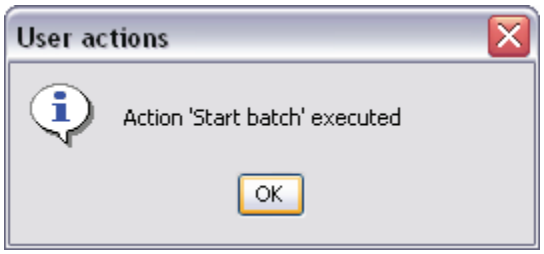
The number and names of the User actions is automatically generated depending on the provider selected during the event mapping phase of the configuration process.

User actions can be invoked by selecting the required action from the context menu associated with a descriptor ('.pad') node displayed under a server file system or by using a command line tool found in the installation directory 'triggerUserAction.bat'.



After the action selection, the IDE will prompt the user to enter the declared parameters of the selected action.

Status message of the action command:



Command line tool usage:
TriggerUserAction.bat <serviceURL> <descriptorName> <actionName> [parameter=value,[parameter=value]...]

Parameter	Sample
ServiceURL	service:jmx:rmi:///jndi/rmi://[hostname]:[port]/ProcessAudit
descriptorName	'NewEmployeeProvisioning'
actionName	'Start batch'
parameter=value	Key=value. If spaces are to be included, enclose the pair in double quotes. Parameters that are not required by the action will be ignored.

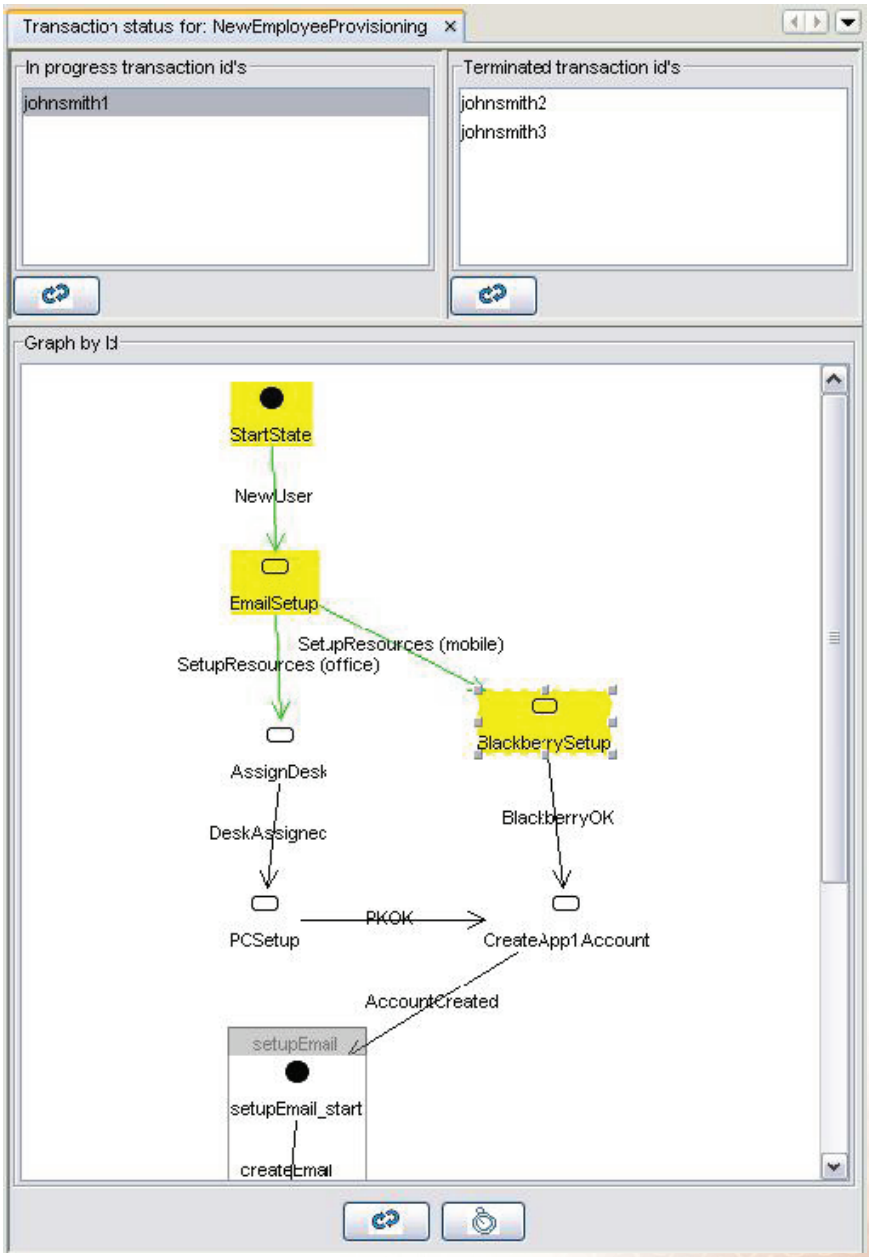
Return codes.

Return codes	Cause/description
0	Action executed
-1	Action failed on the server side. Check server logs.
-2	Failed to prepare arguments or to invoke action.
-3	Failed to validate actionName
-4	Unable to connect to server

To visualize the status of the transactions from a process, select 'Transaction Status' option from the context menu associated with the main descriptor node ('.pad') displayed under a mounted process audit server.

Transaction Status

Transaction status screen:

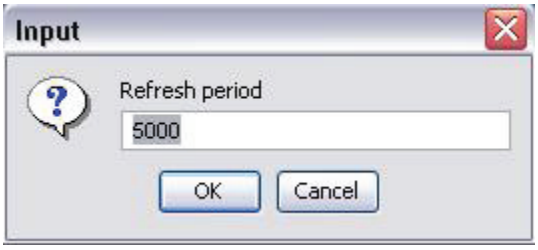


Transaction status panel displays in two different scroll lists the id's of in-progress transaction and id's of terminated transactions. Any selection of an id will determine the graph panel to be updated with the process graph itself.

Each list of id's has a refresh button attached , that will refresh the transaction id's by querying the status of the process.

The graph itself has a refresh button associated, along with an auto-refresh button . A left click on the auto-refresh button will start refreshing the graph by the given refreshing period. Once the auto-refresh was turned on, the button will change its appearance to , and another click on this button will cancel the auto refresh of the graph.

The default value for the refreshing period is 5,000 ms and it can be changed with a right-click on the auto refresh button. A popup dialog will accept the new value for the refreshing period.



Error and exception messages

The table lists the error codes and their possible cause.

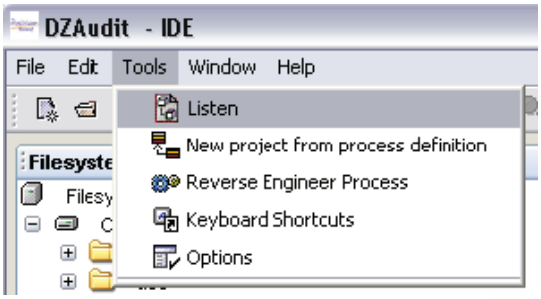
Error code	Cause/description
INTERNAL	Undocumented exception
Failed to capture data: + exception	Invalid provider/ middleware failure/ configuration
The capture file is not valid	File name for the capture file is a

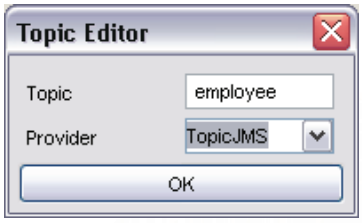
	directory or an invalid file name
Capture file is invalid (not a File)	Check the file name
Unable to process capture file: +exception	File is corrupted or locked by another process.
Restart failed. Check server logs for detailed info.	JMX communication is down. Server was shutdown.
Shutdown scheduling failed. Check server logs for detailed info.	JMX communication is down. Server was stopped.
Failed to connect to server using parameters: + serviceURL	Server is stopped, check communication parameters.
Action 'action name' not executed. Check server logs for detailed info	Server is stopped or in a state incompatible to this command.

Tools

Listen option

To facilitate debugging and understanding of the correlation process, by selecting 'Listen' from the 'Tools' menu, an event set inspection panel will be displayed.





In the 'Topic Editor' window, following fields have to be specified:

- Provider: provider name as declared in 'lookup.xml' to be used to publish the correlated event sets.
- Topic: the topic to publish the messages to. Must be compliant with the conventions of the selected event (middleware) provider.

The usage of the message listening panel is identical to the listening panel described in 'Raw events' section.

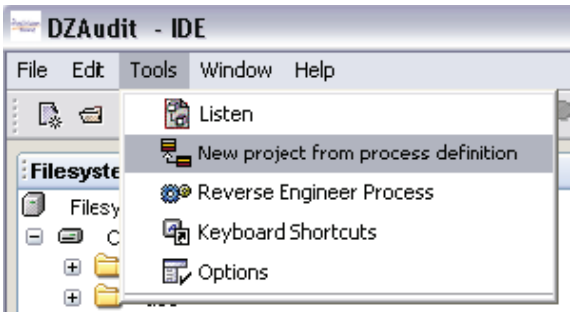
New project from process definition

This wizard automatically creates all the files needed by a new project to be deployed in a server for execution, based on the selection of a business process definition file in UML state chart XMI format.

Following files are created:

- a server configuration file (.pas file).
- a deployment descriptor file (.pad).
- an event mapping file (.pam).

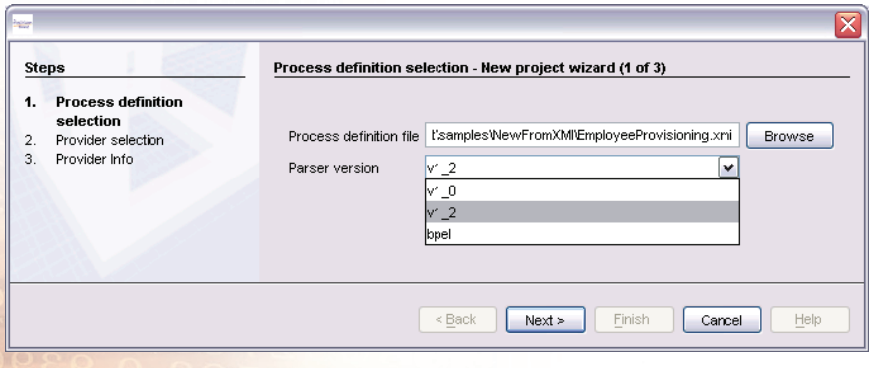
The wizard can be started from 'Tools' menu, 'New project from process definition' option.



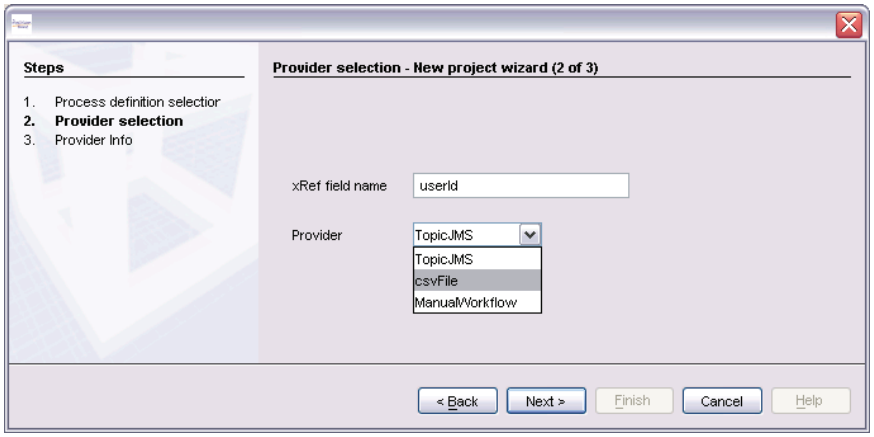
First window accepts a selection of a business process definition file (.xmi file) and a parser version.

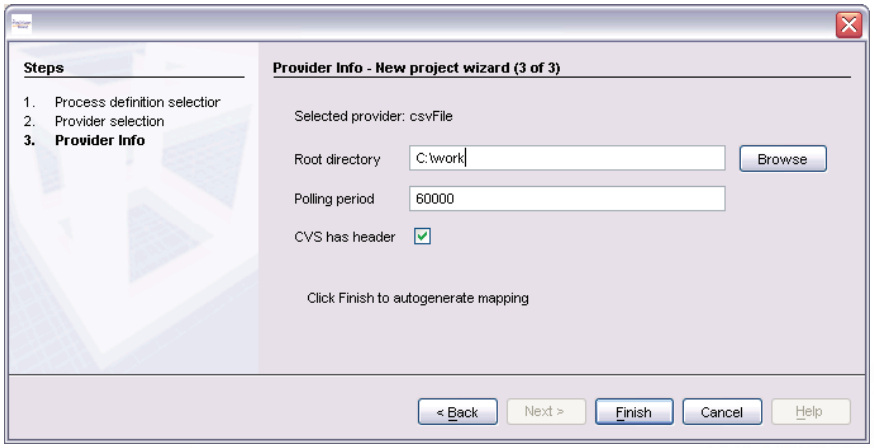
'Browse' button opens a file explorer window that only displays files of .xmi type.

When this button is selected, the descriptor file and the server configuration file are created.

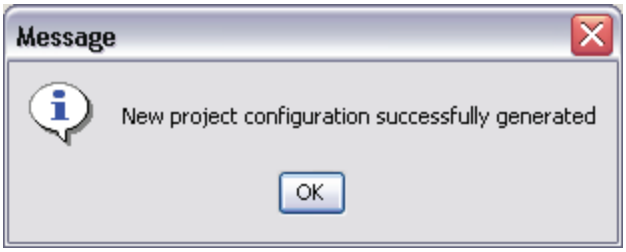


Last two windows of the wizard collect the information needed for the generation of the mapping file (.pam file), as: xRef field name, provider and parameters specific to the provider selected. A detailed description of these two windows can be found in the 'Autogenerate Mapping' section of this documentation.





If the project is successfully generated, a message is displayed. Otherwise, an error message is shown.



Reverse Engineer Process

Reverse engineer process uses advanced data mining techniques to correlate events from a timing and data perspective, extracting the sequence information and guard conditions for a process branches.

It requires a training dataset, whose size is controlled by 'Queue size' parameter, collected in 'Data Input' panel of the wizard.

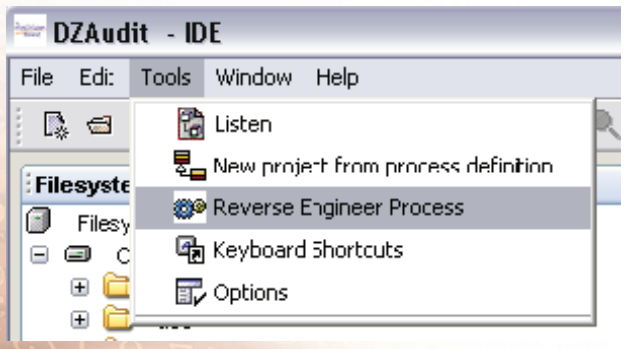
The algorithm will reject all process paths that have a count below the 'Ignore percentage' parameter, whose value has to be entered in 'Data Input' panel of the wizard.

The process to reverse engineer guard conditions is based on decision tree models and might include linear regression formulas.

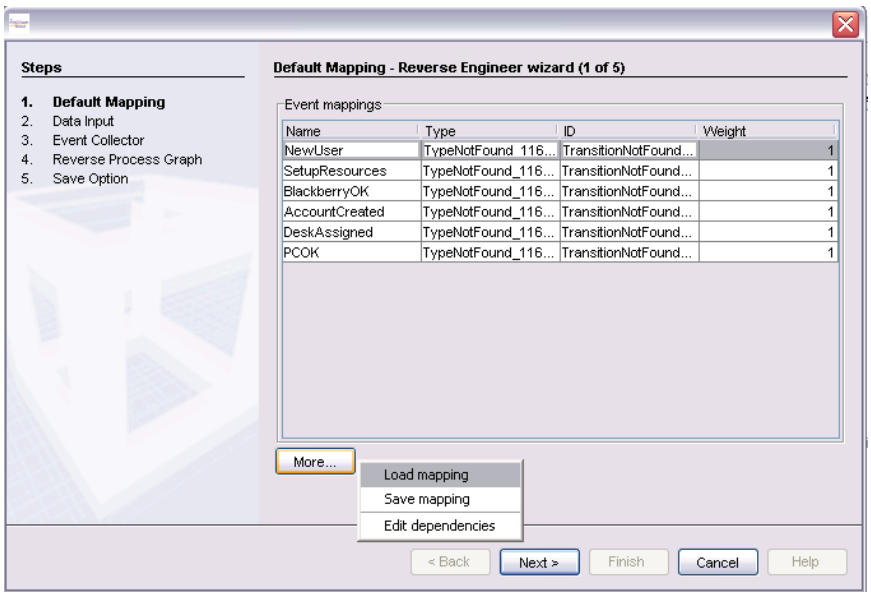
Once the process was reversed, the mapping used to collect the event data can be updated with guard condition information and saved as a .pam file, for direct process audit.

Supported format for the process file is xmi version 1.2.

To start the wizard, from 'Tools' menu select 'Reverse Engineer Process' option.

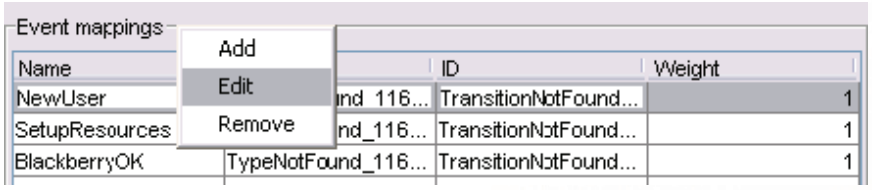


First panel will load an empty mapping to be updated by the user. The mapping is a way to tell the reverse engineering process how to reach the events for capture.



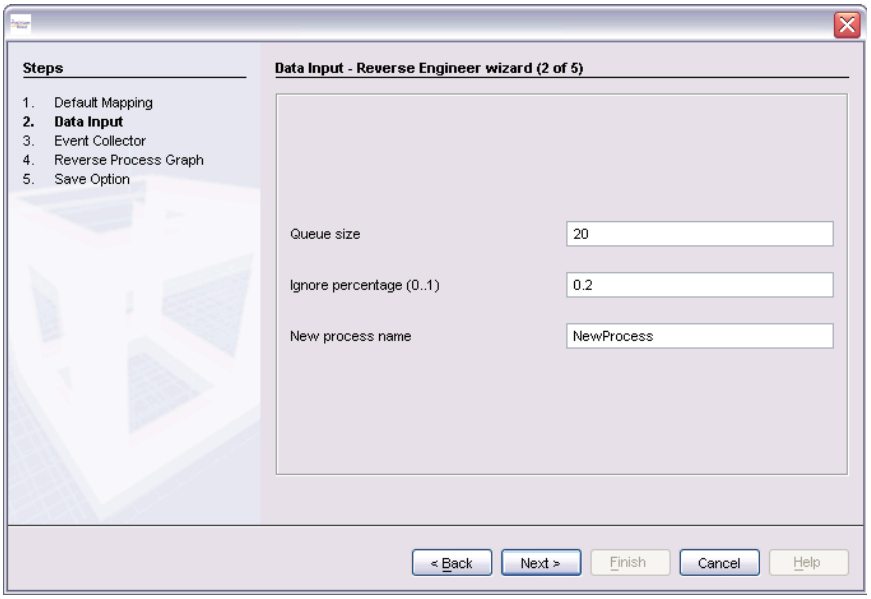
'Load Mapping' option loads an existing mapping.
'Save Mapping' will save the mapping to a selected location.
'Edit Dependencies' option will open dependencies window.

To add/edit/remove, select the corresponding action from the context menu associated with the table/rows.



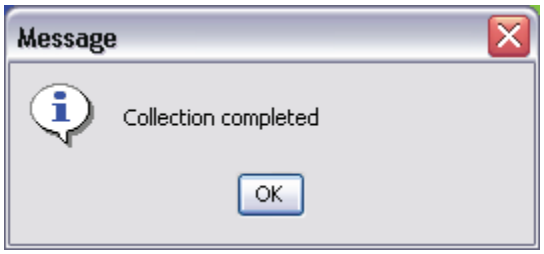
Next panel, 'Data Input' panel, collects additional information needed by the reverse engineering process, such as:

- queue size – number of events to be collected for processing, it also determines the size of the training dataset.
- ignore percentage – all process paths that have a count below the value of this parameter will be rejected by the algorithm.
- new process name – a name for the process to be created.

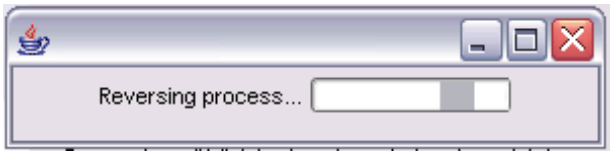


Next step, 'Event Collector' panel is collecting the events and representing the status of the queue.

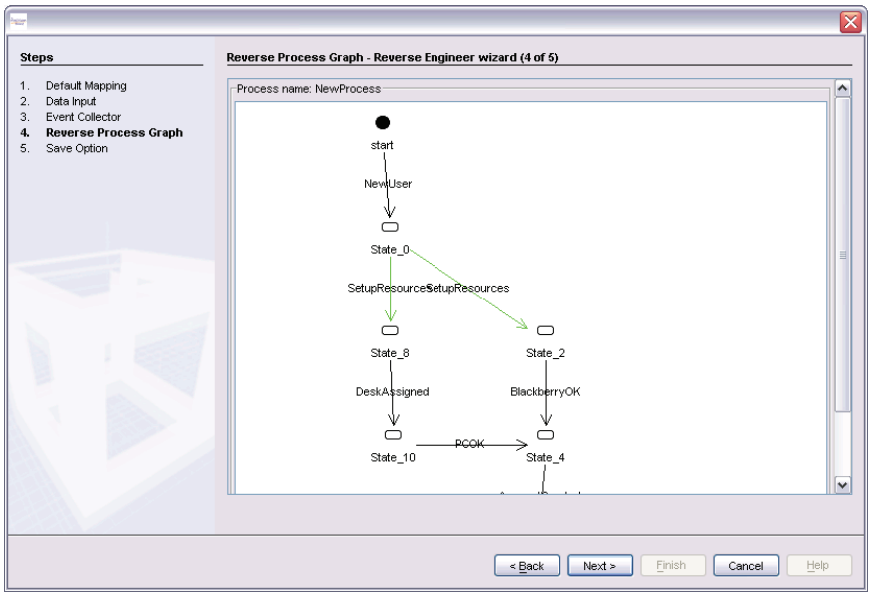
When the count of collected events reaches the queue size, a message will inform the user that the event collection process is completed.



Next step, 'Reverse Process Graph' will start the reverse engineering process. While processing, a progressing bar is displayed.

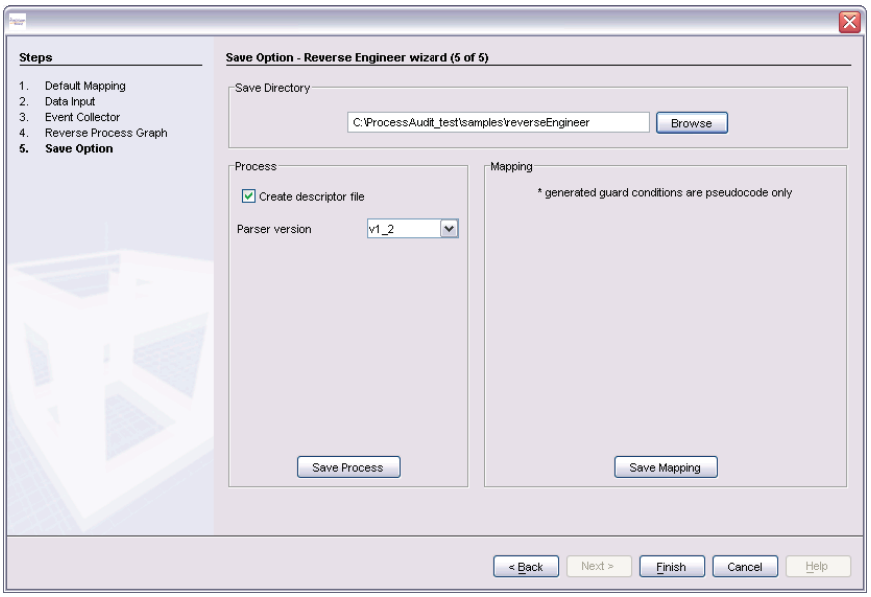


Once the process is completed, the process graph will be displayed.



Last step, 'Save Option' panel, allows the mapping to be saved in the selected directory, along with the descriptor file (.pad) and the process itself (.xmi).

At the initialization of the panel 'Save Process' and 'Save Mapping' buttons are …, but once the saving directory is selected (using ' Browse' option), they become …



Click 'Finish' to exit wizard.

Note: Generated guard conditions for the mapping are pseudo code only.

Event Providers

Introduction

A provider consists of two main components that have a complementary role in sending and receiving of the messages. To facilitate the instance creation of a provider, a factory interface is part of the provider API.

The following interfaces are defined and required for a valid provider implementation:

IBPSignalFactory: holds the common transport layer if possible (used by the producer and listener). Also holds the factory methods for producer and listener instance creation.

IBPEventProducer: sends the supplied event using the native transport mechanisms.

IBPEventListener: notifies the supplied callback whenever a new event instance is available for processing.

A common parameter to the producer and listener components is the 'topic' that translates into the native destination selector (i.e. if the provider is a file based, topic becomes the file name to be used).

Note: *The provider implementation does not require the mandatory implementation of all the interfaces. Please consult the documentation for each specific provider.*

JMS Provider

The JMS provider is one of the providers that should be used for real time events.

The following table lists the parameters needed by the provider, along with their description:

Parameter Name	Description
topic	The JMS message topic (mandatory).
queueSize	The size of the event queue (number of events to be stored in the correlation queue before they are marked as 'expired').
expiryPeriod	Period of time to keep the events in the correlation queue. After an event was marked as 'expired' the correlation process will, most likely, tag it as an anomaly.
java.naming.provider.url	The URL of the JNDI provider. The default value is: 'tcp://localhost:3035'
java.naming.factory.initial	The fully qualified class name of the factory class that will create an initial context. The default value is: *'org.exolab.jms.jndi.InitialContextFactory'*
java.naming.security.principal	The identity of the principal for authenticating the caller to the service. The default value is: *'admin'*

java.naming.security.credentials	The credentials of the principal for authenticating the caller to the service. The default value is: *'admin'*
TopicJMS.PROVIDER_URL	System environment variable that represents the URL of the JNDI provider. If *'java.naming.provider.url'* parameter is present in the configuration of the provider, it will override the value of this variable.
TopicJMS.INITIAL_CONTEXT_FACTORY	System environment variable that represents the fully qualified class name of the factory class that will create an initial context. If *'java.naming. factory.initial'* parameter is present in the configuration of the provider it will override the value of this variable.
TopicJMS.SECURITY_PRINCIPAL	System environment variable; it represents the identity of the principal for authenticating the caller to the service. If *'java.naming.security.principal'* parameter is present in the configuration of the provider, it will override the value of this variable.
TopicJMS.SECURITY_CREDENTIALS	System environment variable, represents the credentials of the principal for authenticating the caller to the service. If *'java.naming.security.credentials'* parameter is present in the configuration of the provider, it will override the value of this variable.

Request Reply JMS Provider

The Request Reply JMS provider is one of the providers that should be used for real time events. The server reply is a message that contains only two fields: VALID (Boolean) true if the message pass validation, and "ERROR_MESSAGE" contains the error message if the message failed validation.

The following table lists the parameters needed by the provider, along with their description:

Parameter Name	Description
topic	The JMS message topic (mandatory).
queueSize	The size of the event queue (number of events to be stored in the correlation queue before they are marked as 'expired').
expiryPeriod	Period of time to keep the events in the correlation queue. After an event was marked as 'expired' the correlation process will, most likely, tag it as an anomaly.
java.naming.provider.url	The URL of the JNDI provider. The default value is: 'tcp://localhost:3035'
java.naming.factory.initial	The fully qualified class name of the factory class that will create an initial context. The default value is: *'org.exolab.jms.jndi.InitialContextFactory'*
java.naming.security.principal	The identity of the principal for authenticating the caller to the service. The default value is:

	'admin'
java.naming.security.credentials	The credentials of the principal for authenticating the caller to the service. The default value is: *'admin'*
TopicJMS.PROVIDER_URL	System environment variable that represents the URL of the JNDI provider. If *'java.naming.provider.url'* parameter is present in the configuration of the provider, it will override the value of this variable.
TopicJMS.INITIAL_CONTEXT_FACTORY	System environment variable that represents the fully qualified class name of the factory class that will create an initial context. If *'java.naming.factory.initial'* parameter is present in the configuration of the provider it will override the value of this variable.
TopicJMS.SECURITY_PRINCIPAL	System environment variable; it represents the identity of the principal for authenticating the caller to the service. If *'java.naming.security.principal'* parameter is present in the configuration of the provider, it will override the value of this variable.
TopicJMS.SECURITY_CREDENTIALS	System environment variable, represents the credentials of the principal for authenticating the caller to the service. If *'java.naming.security.credentials'* parameter is present in the configuration of the provider, it will override the value of this variable.

Rendezvous Provider

'Rendezvous' provider is the technology adapter used for TIBCO Rendezvous events.

The following table lists the parameters needed by the provider, along with their description:

Parameter Name	Description
topic	The TIBCO Rendezvous message subject (mandatory).
queueSize	The size of the event queue (number of events to be stored in the correlation queue before are marked as 'expired').
expiryPeriod	Period of time to keep the events in the correlation queue. After an event was marked as 'expired' the correlation process will, most likely, tag it as an anomaly.
rvService	System environment variable, represents Tibco rendezvous transport *'service'* parameter.
rvNetwork	System environment variable that represents Tibco rendezvous transport *'network'* parameter.
rvDaemon	System environment variable that represents Tibco rendezvous transport *'deamon'* parameter.

CSV File Provider

The purpose of this provider is to read the content of a .csv file and create events to be pushed into the DZAudit engine.

The usage of the provider depends on the existence of the following directory structure:

rootDir+'/'+mappingNameDir+'/'+topic

If the directory structure exists, it will be checked periodically for new .csv files. 'pollingPeriod' parameter specifies the interval for polling. When a new .csv file is found, the engine will start processing it by pushing events.

When the processing of the file ends, the file will be renamed to:

<file_name>.csv.processed_<timestamp>,
where <timestamp> represents the timestamp when the processing ends, in the 'dd_MMM_yy_HH_mm_ss_SSS' format. For more information about date formats, please consult Java API for Simple DateFormat.

Note: *CSV stands for Comma Separated Values.*

To enable the usage of the provider in the IDE, the lookup file

('lookup.xml') should include the following information:

```
<provider name="csvFile" type="in">
<class>com.decisionzone.businessbroker.event.csvFile.
CsvFileSignalFactory</class>
<parameters>
<parameter name="topic" type="in">
<className>java.lang.String</className>
<defaultValue>csvFile_default_topic</defaultValue>
</parameter>
<parameter name="rootDir" type="in">
```

```
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="mappingNameDir" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="pollingPeriod" type="in">
<className>java.lang.Long</className>
<defaultValue>5000</defaultValue>
</parameter>
<parameter name="csvHasHeader" type="in">
<className>java.lang.Boolean</className>
<defaultValue>true</defaultValue>
</parameter>
<parameter name="timestampField" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="dateFormat" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
</parameters>
</provider>
```

Also, the resourceMapping.xml file, under <signalEventTypes> should have an entry like the following:

```
<eventType name="csvFile" provider="com.decisionzone.businessbroker.
event.csvFile.CsvFileSignalFactory" active="true"/>
```

The following table lists the parameters needed by the provider, along with their description:

Parameter Name	Description
topic	Topic name, usually corresponds to the event name.
rootDir	Root directory.
mappingNameDir	Mapping name directory.
pollingPeriod	Specifies the polling period for checking for new .csv files.
cvsHasHeader	A 'true' value specifies that the first line of the .csv file represents the header description. A 'false' value indicates that the header is not included in the .csv file, and an additional file with .meta extension is provided in the same directory where the .csv file will be dumped. The .meta file will contain a single line of comma separated values that will represent the header associated with the .csv file.
timestampField	Specifies a column name from the .csv file header or .meta file that represents the timestamp.
dateFormat	If a timestampField is provided, a date format can be specified as well, and it represents the format of the timestamp value (e.g. 'dd/MM/yyyy'; for more information please consult Java API for SimpleDateFormat)

JDBC Provider

JDBC provider enables connectivity to the database.

To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

```
<provider name="JDBC" type="in">
<class>com.decisionzone.businessbroker.event.jdbc.
BPSignalFactory</class>
<parameters>
<parameter name="topic" type="in">
<className>java.lang.String</className>
<defaultValue>topic</defaultValue>
</parameter>
<parameter name="jdbcClassName" type="in">
<className>java.lang.String</className>
```

```
<defaultValue>oracle.jdbc.driver.OracleDriver</defaultValue>
</parameter>
<parameter name="url" type="in">
<className>java.lang.String</className>
<defaultValue>jdbc:oracle:thin:@&lt;host&gt;:1521:&lt;sid&gt;
</defaultValue>
</parameter>
<parameter name="user" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="password" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="selectSQL" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="indexFields" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="alternateFieldNames" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="timestampField" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="eventQueueClassName" type="in">
<className>java.lang.String</className>
<defaultValue>com.decisionzone.pattern.xmi.
engine.JDBCQueue</defaultValue>
</parameter>
<parameter name="tempTableName" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
</parameters>
</provider>
```

The table lists the parameters needed by the provider, along with their description:

Parameter Name	Description
topic	Topic name
jdbcClassName	JDBC driver class name to be used for the database connection.
url	Database url (driver specific connection string).
user	User name to be used for database authentication.
password	User's password.
selectSQL	A valid select sql statement that returns the columns to be used as event fields.
indexFields	Comma separated values that represent index fields of the table.
alternateFieldNames	Comma separated values that overwrite the field names returned by the select statement.
timestampField	Specifies the timestamp field.
eventQueueClassName	Implementation class name of the correlation queue. Default value is: 'com.decisionzone.pattern.xmi.engine.JDBCQueue'
tempTableName	During the runtime phase, while events are pushed into the engine, this table will be used to mark the events that have been processed. At engine restart, the processing will continue with the first row that has not been marked as processed. If this temporary table is emptied, the processing will start from the beginning.

Manual Workflow Provider

Manual Workflow provider will transform manual user inputs into events that are sent to the DZAudit engine.

To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

```
<provider name="ManualWorkflow" type="in">
<class>com.decisionzone.businessbroker.event.
manualWorkflow.ManualWorkflowSignalFactory</class>
<parameters>
<parameter name="TransportProvider" type="in">
<className>java.lang.String</className>
<defaultValue>TopicJMS</defaultValue>
</parameter>
<parameter name="Role" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
</parameters>
</provider>
```

Parameter Name	Description
TransportProvider	Transport provider (e.g. 'TopicJMS')
Role	Role value, needed after deploying DZAuditManualWorkflow. Specifies the role assigned to the user that will input the data through the Manual Workflow UI.

Manual Workflow Provider (Enforcement)

Manual Workflow provider will transform manual user inputs into events that are sent to the DZAudit engine. The engine will validate data and notify the user if the validation failed. A validator must be configured.

To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

```
<provider name="RRManualWorkflow" type="in">
<class>com.decisionzone.businessbroker.event.
RRManualWorkflow.RRManualWorkflowSignalFactory</class>
<parameters>
<parameter name="TransportProvider" type="in">
<className>java.lang.String</className>
<defaultValue>TopicJMS</defaultValue>
</parameter>
<parameter name="Role" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
</parameters>
</provider>
```

Parameter Name	Description
TransportProvider	Transport provider (e.g. 'TopicJMS')
Role	Role value, needed after deploying DZAuditManualWorkflow. Specifies the role assigned to the user that will input the data through the Manual Workflow UI.

Email Provider

The email provider allows the engine to listen to incoming email messages, process them and push events to the DZAudit engine based on the information collected from the email. It also provides the means to send emails whenever an anomaly occurs.

In order to enable the usage of the email provider into the IDE, lookup.xml file needs an entry like the one below:

```
<provider name="eMail" type="in/out">
<class>com.decisionzone.businessbroker.event.email.
BPEmailSignalFactory</class>
<parameters>
<parameter name="topic" type="in/out">
<className>java.lang.String</className>
<defaultValue>topic</defaultValue>
</parameter>
<parameter name="malServer" type="in/out">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="userName" type="in/out">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="password" type="in/out">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
```

```
<parameter name="fromRegex" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="fromFields" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="toRegex" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="toFields" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="ccRegex" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="ccFields" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="bccRegex" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="bccFields" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="subjectRegex" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="subjectFields" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="contentRegex" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="contentFields" type="in">
```

<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="xslMailTemplate" type="out">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="title" type="out">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="minimumInterval" type="out">
<className>java.lang.Long</className>
<defaultValue>1000</defaultValue>
</parameter>
</parameters>
</provider>
resourceMapping.xml should have specified an entry like the following:

<eventType name="eMail"
provider="com.decisionzone.businessbroker.event.email.BPEmailSignal
Factory" active="true"/>

Parameters description:

Parameter Name	Description
topic	Topic name.
mailServer	Mail server name.
userName	Username for the email account.
password	Password for mail authentication.
fromRegex	Regular expression to parse the email's 'from' field.
fromFields	Comma separated event fields that are populated with the values returned by parsing 'from' field.
toRegex	Regular expression to parse the email's 'to' field.
toFields	Comma separated event fields that are populated with the values returned by parsing 'to field.
ccRegex	Regular expression to parse the email's 'cc' field.
ccFields	Comma separated event fields that are populated with the values returned by parsing 'cc' field.

bccRegex	Regular expression to parse the email's 'bcc' field.
bccFields	Comma separated event fields that are populated with the values returned by parsing 'bcc' field.
subjectRegex	Regular expression to parse the email's 'subject' field.
subjectFields	Comma separated event fields that are populated with the values returned by parsing 'subject' field.
contentRegex	Regular expression to parse the email's 'content' field.
contentFields	Comma separated event fields that are populated with the values returned by parsing 'content' field.
xslMailTemplate	Name of a template applied to format the content of the email. Includes the path of the template, too. Parameter is needed only by the producer.
title	Process name. Parameter is needed only by the producer.

Queue JMS Provider

Provider's entry in 'resourceMapping.xml':

<eventType name="QueueJMS"
provider="com.decisionzone.businessbroker.event.queueJMS.Queue
JMSSignalFactory" active="true"/>

Entry to be specified in the lookup file:

<provider name="QueueJMS" type="in/out">
<class>com.decisionzone.businessbroker.event.queueJMS.
QueueJMSSignalFactory</class>
<parameters>
<parameter name="queueName" type="in/out">

```
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="queueSize" type="in">
<className>java.lang.Integer</className>
<defaultValue>10</defaultValue>
</parameter>
<parameter name="expiryPeriod" type="in">
<className>java.lang.Long</className>
<defaultValue>10000</defaultValue>
</parameter>
<parameter name="checkForceExpired" type="in">
<className>java.lang.Boolean</className>
<defaultValue>false</defaultValue>
</parameter>
</parameters>
</provider>
```

Parameters description:

Parameter Name	Description
queueName	Queue name.
queueSize	Queue size.
expiryPeriod	Expiry period (in ms).

WS-Push Provider

WS-Push provider allows the engine to expose a web-service that external clients can use to push events directly into the DZAudit engine. On the outbound side, it allows the DZAudit engine to directly call external web-services.

To register the provider into the IDE and DZAudit server, the following provider entry needs to be added in 'resourceMapping.xml':

<eventType name="webServices_push" provider="com.decisionzone.businessbroker.event.wspush.WSPushSignalFactory" active="true"/>
To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

<provider name="webServices_push" type="in/out"> <class>com.decisionzone.businessbroker.event.wspush.WSPushSignalFactory</class>
<parameters>
<parameter name="topic" type="in">
<className>java.lang.String</className>
<defaultValue>ws_default_topic</defaultValue>
</parameter>
</parameters>
</provider>

Description of parameters used by WS-push provider inbound section:

Parameter Name	Description
serviceId	Unique identifier(String) required for each event type provider instance. This parameter is required
port	Port on which the embedded server will listen for web-service HTTP requests. (IP port number)
axis2ConfigurationFileName	The name of the file containing the axis configuration. See Axis2 documentation

	regarding the schema and valid configurations.
serviceName	The name of the service to be published. Default: 'Push'
pushImplClassName	FQ name of the class to handle the web-service invocations.

For the inbound section, once configured and the server is started, the generated WSDL can be retrieved at the following URL:
http://localhost:[port]/dzaudit/wspush/[ServiceName]?wsdl

Ex: http://localhost:18080/dzaudit/wspush/Push?wsdl

Description of parameters used by WS-push provider outbound section:

Parameter Name	Description
wsdlURL	The URL of the WSDL describing the web-service to be invoked.
portName	The name of the port from the wsdl to be used.
operationName	The name of the operation from the selected port to be invoked.
mappingURL	The URL of the parameter mapping file. See later sections for details on the mapping file.
mappingCode	If mapping URL is not specified, then this field will be used to pass the XML mapping to the provider.

packageName	Name of the package to be used for the auto-generated invocation code. If is not specified, it defaults to: 'wspush.default'.
buildDirectory	The name of the directory where the generated code and the compiled classes will be stored. If is not specified, it defaults to: 'wsBuild'.
forceRebuild	Boolean value. If 'true' it will force the overwrite of all the generated files. If not specified, it defaults to 'true'

The schema of the parameter mapping specified a list of entries with the following attributes:

- name : the name of the parameter from the invoked web-service
- description: short description for documentation purposes.
- parameterIndex: the index of the parameter
- content-String.: a JavaScript expression that returns the value to be passed during the web-service invocation.

The following example entry maps the first parameter of the invoked web-service to the value of the field 'a' from the internal BPEvent, converted to a float number.

<Mapping name="a" description="NA" parameterIndex="0">event.
getProperty('a').floatValue()</Mapping>
WSOperationParameterMapping.xsd
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="ParameterMapping">
<xs:complexType>
<xs:sequence>

```
<xs:element name="Mapping" type="MappingType" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name="MappingType">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="name" type="xs:string" use="optional" default="NA"/>
<xs:attribute name="description" type="xs:string" use="optional"
default="NA"/>
<xs:attribute name="parameterIndex" type="xs:int" use="required"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:schema>
```

WS Provider

WS provider allows the engine to non-intrusively intercept web-service invocations off the 'wire' without the introduction of any proxy or tunnel in the communication path.

The payload of the web-service invocations is reassembled from low-level Ethernet packets captured in real-time. These packets are processed by a multi-layer protocol decoder (TCP/HTTP/WS) and the result represents the original payload of the web service invocation (parameters or response) un-marshaled to java objects.

Note: *This provider is an 'inbound only' type.*

To register the provider into the IDE and DZAudit server, the following provider entry needs to be added in 'resourceMapping.xml':

<eventType name="WebServices" provider="com.decisionzone.business-broker.event.ws.WSSignalFactory" active="true"/>

To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

<provider name="webServices" type="in"> <class>com.decisionzone.busi-nessbroker.event.ws.WSSignalFactory</class>
<parameters>
<parameter name="topic" type="in">
<className>java.lang.String</className> <defaultValue>ws_default_top-ic</defaultValue>
</parameter>
</parameters>
</provider>

Description of parameters used by WS provider:

Parameter Name	Description
wsdlURL	The URL of the WSDL describing the web-service to be invoked.
operationName	The name of the operation from the selected port to be invoked.
packageName	Name of the package to be used for the auto-generated invocation code. If is not specified, it defaults to: 'wspush.default'.
buildDirectory	The name of the directory where the generated code and the compiled classes will be stored. If is not specified, it defaults to: 'wsBuild'.
forceRebuild	Boolean value. If 'true' it will force the overwrite of all the generated files. If not specified, it defaults to 'true'

captureRequest	Boolean value. If 'true' the provider will capture the request side of the web-service invocation.
captureResponse	Boolean value. If 'true' the provider will capture the response side of the web-service invocation.
deviceName	The name of the device used to capture the low-level Ethernet packets. During the configuration, the user can select the device name from a drop-down list of 'human-readable' device names.
captureSize	Integer value. It controls the size of the capture window (number of packets to be kept in memory at any point in time).
filter	String value representing the filtering to be performed before the packets reach the packet queue. For details on syntax, see WinPCap documentation.
findPeriod	Integer value. It represent the period of time to wait between the processing cycles of the packet queue.
inactiveClearTimeout	Integer value. Specifies the time in milliseconds that a communication packet flow has no new data before is marked as inactive and all associated packets are removed from the event queue.

SMTP Provider

WS provider allows the engine to non-intrusively intercept SMTP communication between a client and a server off the 'wire' without the introduction of any proxy or tunnel in the communication path.

The payload of the SMTP messaged is reassembled from low-level Ethernet packets captured in real-time. These packets are processed by a multi-layer protocol decoder (TCP/HTTP/SMTP) and the result represents the original payload of the transferred mail message.

Note: *This provider is an 'inbound only' type. Attachments are not supported.*

To register the provider into the IDE and DZAudit server, the following provider entry needs to be added in 'resourceMapping.xml':

<eventType name="SMTP" provider="com.decisionzone.businessbroker.event.ws.SMTPSignalFactory" active="true"/>

To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

<provider name="SMTP" type="in"> <class>com.decisionzone.businessbroker.event.ws.SMTPSignalFactory</class>
<parameters>
<parameter name="topic" type="in">
<className>java.lang.String</className> <defaultValue>smtp_default_topic</defaultValue>
</parameter>
</parameters>
</provider>

Description of parameters used by SMTP provider:

Parameter Name	Description
contentRegex	Regex expression that must contain a capturing group for every property name listed in the 'propertiesNames' parameter. As a reference to the Employee Provisioning sample package, for 'SetupResourcesForOffice' event, this parameter should have the following value: ^([^,\n]*),([^,\n]*)$
contentFields	Comma separated values for property names. As a reference to the Employee Provisioning sample package, for 'SetupResourcesForOffice' event, this parameter should have the following value: userId,isMobile
packageName	Name of the package to be used for the auto-generated invocation code. If is not specified, it defaults to: 'wspush.default'.
buildDirectory	The name of the directory where the generated code and the compiled classes will be stored. If is not specified, it defaults to: 'wsBuild'.
forceRebuild	Boolean value. If 'true' it will force the overwrite of all the generated files. If not specified, it defaults to 'true'
deviceName	The name of the device used to capture the low-level Ethernet packets. During the configuration, the user can select the device name from a drop-down list of 'human-readable' device names.
captureSize	Integer value. It controls the size of the capture window (number of packets to be kept in memory at any point in time).
filter	String value representing the filtering to be performed before the packets reach the packet queue. For details on syntax, see WinPCap documentation.

Text File Provider

Text provider allows the engine to poll the content of a text file, process it and push events to the DZAudit engine based on the information collected from the file. It also provides the means to write the content of the events to a specified file.

Provider's entry in 'resourceMapping.xml':

<eventType name="File"
provider="com.decisionzone.businessbroker.event.text.BPTextSignalFactory" active="true"/>

To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

```
<provider name="File" type="in/out">
<class>com.decisionzone.businessbroker.event.text.BPTextSignalFactory
</class>
<parameters>
<parameter name="topic" type="in/out">
<className>java.lang.String</className>
<defaultValue>topic</defaultValue>
</parameter>
<parameter name="textFileName" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="regex" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="propertiesNames" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="pollPeriod" type="in">
<className>java.lang.Long</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="minimumInterval" type="out">
<className>java.lang.Long</className>
<defaultValue>1000</defaultValue>
</parameter>
</parameters>
</provider>
```

Description of parameters used by text file provider:

Parameter Name	Description
textFileName	Name of the text file that is being polled.
regex	Regex expression that must contain a capturing group for every property name listed in the 'propertiesNames' parameter. As a reference to the Employee Provisioning sample package, for 'SetupResourcesForOffice' event, this parameter should have the following value: ^([^,\n]*),([^,\n]*)$
propertiesNames	Comma separated values for property names. As a reference to the Employee Provisioning sample package, for 'SetupResourcesForOffice' event, this parameter should have the following value: userId,isMobile
pollPeriod	Polling period (in ms) that determines how often 'textFileName' is being read.
xslTemplate	Used only by the provider's producer, 'xslTemplate' represents the name of an .xsl file that will determine the output structure of the 'textFileName' file. If no .xsl template is defined, the output of the text file represents a concatenation of xml structures. If an .xsl template is specified, the output of the text file will be determined by the template itself. A sample .xsl file can be found in "EmployeeProvisioningFile" sample directory.

Script Provider

The Script provider enable execution of javascript scriptlets.

The output section of the provider (implementation of IBPEvent Producer interface) is used to trigger actions instead of publishing an event. The actions are implemented as a javascript scriptlet. The action scriptlet must implement the send(event) fiunction:

function send(event) { <your code here>}
The input section of the provider (implementation of IBPEventListener interface) starts the execution of the javascript scriptlet, on a different thread with a 1 second delay from the provider initialization. The provider will insert into the execution context of the scriptlet the following variables:

- topic – the value of the topic parameter of the provider
- xRef – the cross reference object provided by the engine
- callback – the callback object, an object that implement ScriptCallback interface.
- event – an empty event object that implements IBPEvent interface

The javascript scriptlet must call the onEvent(IBPEvent event) method of the callback object, to notify the provider listeners about the desired event.

To enable the usage of the provider in the IDE, the lookup file ('lookup.xml') should include the following information:

<provider name="ScriptAction" type="in/out">
<class>com.decisionzone.businessbroker.event.scriptAction.ScriptActionSignalFactory</class>
<parameters>
<parameter name="topic" type="in">
<className>java.lang.String</className>
<defaultValue>topic</defaultValue>

```
</parameter>
<parameter name="inScriptCode" type="in">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
<parameter name="outScriptCode" type="out">
<className>java.lang.String</className>
<defaultValue></defaultValue>
</parameter>
</parameters>
</provider>
```

Description of parameters used by scriptAvtion provider:

Parameter Name	Description
topic	Topic name.
outScript	The javascript scriptlet used by the output section of the provider
inScript	The javascript scriptlet used by the input section of the provider

Additional features

DZAudit Bridge

DZAudit Bridge is a utility tool that listens to events sent by the process audit server, acquires statistical data and aggregates it into one or multiple tables, depending on the content of the configuration file ('bridgeConfig.xml'). Data stored in the aggregate table(s) can be visualized in the Dashboard server.

Working directory for DZAudit Bridge is 'AuditBridge'.

To start the utility tool, launch 'startProcessBridge.bat' from the working directory.

Bridge configuration file

DZAudit Bridge's behavior is determined by the specifications given in the configuration file ('bridgeConfig.xml'). The schema of the configuration file is presented in Appendix C.

The following list represents mandatory properties describing the listener and database connection:
- topic – topic name; same as the one specified for publishing of the event sets resulted from the correlation process (please see 'Raw Events' section).
- provider – provider name; same as the one specified in publishing of the event sets resulted from the correlation process (please see 'Raw Events' section).
- jdbcDriver – jdbc driver (for Oracle, use 'oracle.jdbc.driver.OracleDriver').

- connectionURL – connection URL string. For Oracle, use 'jdbc:oracle:thin:@<server>:1521:<sid>', where
<server> = oracle server name
<sid> = sid of the oracle server
- user – database user name (PROCESS_AUDIT).
- password – database user password (PROCESS_AUDIT).

Note: *If any of the properties mentioned above is not specified or the configuration file is not found, an exception will occur.*

Note: *The database for the Dashboard server requires the existence of a user with 'PROCESS_AUDIT' as username and password.*

Sample for mandatory properties entries in the configuration file:

<listener topic="employee" provider="TopicJMS"/>
<dbConnection jdbcDriver="oracle.jdbc.driver.OracleDriver" connectionURL="jdbc:oracle:thin:@prod01:1521:nmi" user="PROCESS_AUDIT" password="PROCESS_AUDIT"/>

Aggregate table(s)

The functionality of DZAudit Bridge is given by the content of the configuration file, by the <eventMapping> entry.

1. <eventMapping> is empty

The aggregate table that keeps the statistical data is 'DZ_A_E_AGG'.

If this table does not exist in 'PROCESS_AUDIT' database user, it is automatically created.

Following table represents the structure of the aggregate table:

Field name	Field description
PROCESS_NAME	process name
PATH	process path
EVENT_NAME	event name
TS	timestamp
ANOMALY_COUNT	number of events with anomalies (sum of ANOMALY_COUNT, EXPIRED_COUNT, MISSING_COUNT and GC_NOT_MET_COUNT)
MISSING_COUNT	number of events with anomalies of missing source
EXPIRED_COUNT	number of events with anomalies detected for expired events
GC_NOT_MET_COUNT	number of events with anomalies of guard condition not met
NORMAL_COUNT	number of events with no anomalies

2. <eventMapping> not empty

If the <eventMapping> is not empty, every entry of this type must have a corresponding table in the database, capturing only that event's data.

At bridge's initialization, if the corresponding tables are not present in the database, an .sql file is generated, containing the sql statements to create the tables needed, then the application will quit, giving the user the opportunity to create the tables needed.

Generated .sql file is placed in the root directory of DZAudit Bridge, with the following name:

createTable_<currentMillisecond>.sql,
where <currentMillisecond> represents the current millisecond in a 'dd_MMM_yy_HH_mm_ss_SSS' format (for more information about date formats, please consult Java API for SimpleDateFormat).

Use the content of the .sql file to create the table(s) in your database before using the bridge tool.

Attributes for <eventMapping> entry:

- name – represents the event mapping, as defined in the mapping.
- tableName – the name of the corresponding table in the database.

Attributes for <fieldMapping> entry:

- name – event field name, as defined in the mapping.
- tableFieldName – table column name.
- tableFieldType – table column type, by Oracle specifications.
- tableFieldLength – length of the table column.
- eventFieldType – type of the event field. Acceptable values are: 'java-lang.String', 'java.lang.Number', 'java.lang.Boolean' and 'java.util.Date'.
- dateFormat – specifies the date format.

All attributes except <tableFieldLength> and <dateFormat> are mandatory.

After all tables are created in the database, DZAudit Bridge is ready to listen to events, save the events data for further analysis using the Dashboard server.

Dashboard hierarchies

For a multidimensional visualization of the data stored in the aggregate table, Dashboard server product has to be installed. Several hierarchies have been defined to facilitate the data analysis.

Dashboard server uses Mondrian as an OLAP (Online Analytical Processing) server and JPivot as an OLAP client. Mondrian OLAP server executes queries written in the MDX language (multi-dimensional expressions) and presents the result in a multidimensional format.

The working directory of DZAudit Bridge contains 2 files that define the hierarchies:

- 'hierarchy.properties'
- 'dashboardMondrian.xml'

The content of these to files are to be appended to the content of the corresponding files in the Dashboard server.

The property file ('hierarchy.properties') holds information regarding connection to the database, hierarchy name and title, cube and dimension where each hierarchy is defined, MDX query of each hierarchy, and hierarchy structure, determined by its levels. See Appendix A for property file content.

Schema definition file ('dashboardMondrian.xml') contains the schema that defines the mapping between the logical model and the physical model.

The logical model consists of cubes, dimension, hierarchies, levels and members. The physical model is the source of the data that is presented through the logical model. For more information regarding Mondrian schemas, please consult Mondrian documentation.

See Appendix B for the content of the schema file.

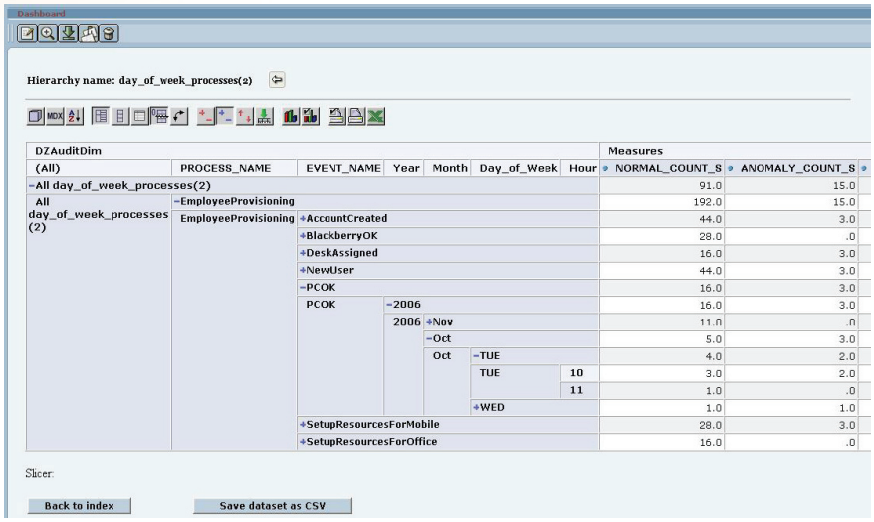
Measures defined in the Mondrian schema use two kinds of aggregators:

- sum (measure_name=<table_column_name>_S)
- average /avg (measure_name=<table_column_name>_A)

Four hierarchies have been defined, as follows:

- daily_processes(1) – groups events daily, by hour. Hierarchy structure: PROCESS_NAME, PATH, EVENT_NAME, Year, Month, Day_of_Month, Hour.
- daily_processes(2) – groups events daily, by hour, without PATH level. Hierarchy structure: PROCESS_NAME, EVENT_NAME, Year, Month, Day_of_Month, Hour.
- day_of_week_processes(1) – groups events by day of week, by hour. Hierarchy structure: PROCESS_NAME, PATH, EVENT_NAME, Year, Month, Day_of_Week, Hour.
- day_of_week_processes(2) - groups events by day of week, by hour, without PATH level. Hierarchy structure: PROCESS_NAME, EVENT_NAME, Year, Month, Day_of_Week, Hour.

							Measures	
PROCESS_NAME	PATH	EVENT_NAME	Year	Month	Day	Hour	NORMAL_COUNT_S	ANOMALY_COUNT_S
sses(1)							91.0	15.0
−EmployeeProvisioning							91.0	15.0
EmployeeProvisioning	+AccountCreated,BlackberryOK,NewUser,SetupResourcesForMobile						24.0	.0
	−AccountCreated,DeskAssigned,PCOK						.0	9.0
	AccountCreated,DeskAssigned,PCOK	−AccountCreated					.0	3.0
		AccountCreated	−2006				.0	3.0
			2006	−Oct			.0	3.0
				Oct	−03		.0	2.0
					03	10	.0	2.0
					−04		.0	1.0
					04	08	.0	1.0
		+DeskAssigned					.0	3.0
		+PCOK					.0	3.0
	+AccountCreated,DeskAssigned,PCOK,SetupResourcesForOffice,NewUser						35.0	.0
	+AccountCreated,SetupResourcesForOffice,DeskAssigned,PCOK,NewUser						20.0	.0
	+NewUser						.0	1.0
	+SetupResourcesForMobile						.0	1.0
	+SetupResourcesForMobile,BlackberryOK,AccountCreated,NewUser						12.0	.0
	+SetupResourcesForMobile,NewUser						.0	4.0



Error and exception messages

Following table lists possible error codes.

Error code	Cause/description
Bridge configuration file not found	Bridge configuration file 'bridgeConfig.xml' not found.
Provider value not found	Provider value not found in 'bridgeConfig.xml' configuration file.
driverClassName not found	Value for driverClassName not found in 'bridgeConfig.xml' file.
connectionURL not found	Value for connectionURL not found in bridge configuration file.
User authentication not found	Values for database authentication (user or password) not found in bridge properties file.

Appendix A

Hierarchies property file (hierarchies.properties)

```
#Hierarchies
jdbcDriver=oracle.jdbc.driver.OracleDriver
jdbcUrl=jdbc:oracle:thin:MONDRIAN_DASHBOARD/MONDRIAN_
DASHBOARD@<SERVER>:1521:<SID>
catalogUri=/WEB-INF/queries/dashboardMondrian.xml
hierarchyList=divider_DZAudit,daily_processes(1),daily_
processes(2),day_of_week_processes(1),day_of_week_processes(2)
name_divider_DZAudit=DZAudit
divider_divider_DZAudit=true
name_daily_processes(1)=daily_processes(1)
title_daily_processes(1)=daily_processes(1)
cube_daily_processes(1)=DZAuditCube
dimension_daily_processes(1)=DZAuditDim
query_daily_processes(1)=select NON EMPTY {[Measures].[NOR-
MAL_COUNT_S],[Measures].[ANOMALY_COUNT_S],[Measures].
[EXPIRED_COUNT_S],[Measures].[MISSING_COUNT_S],[Measures].
[GC_NOT_MET_COUNT_S],[Measures].[NORMAL_COUNT_A],[Measures].
[ANOMALY_COUNT_A],[Measures].[EXPIRED_COUNT_A],[Measures].
[MISSING_COUNT_A],[Measures].[GC_NOT_MET_COUNT_A]} ON col-
umns, NON EMPTY [DZAuditDim.daily_processes(1)].[All daily_process-
es(1)]} ON rows from [DZAuditCube]
hierarchy_daily_processes(1)=(All),PROCESS_NAME,PATH,EVENT_
NAME,Year,Month,Day,Hour,
name_daily_processes(2)=daily_processes(2)
title_daily_processes(2)=daily_processes(2)
cube_daily_processes(2)=DZAuditCube
dimension_daily_processes(2)=DZAuditDim
query_daily_processes(2)=select NON EMPTY {[Measures].[NOR-
MAL_COUNT_S],[Measures].[ANOMALY_COUNT_S],[Measures].
[EXPIRED_COUNT_S],[Measures].[MISSING_COUNT_S],[Measures].
[GC_NOT_MET_COUNT_S],[Measures].[NORMAL_COUNT_A],[Measures].
[ANOMALY_COUNT_A],[Measures].[EXPIRED_COUNT_A],[Measures].
[MISSING_COUNT_A],[Measures].[GC_NOT_MET_COUNT_A]} ON col-
umns, NON EMPTY [DZAuditDim.daily_processes(2)].[All daily_process-
es(2)]} ON rows from [DZAuditCube]
hierarchy_daily_processes(2)=(All),PROCESS_NAME,EVENT_
NAME,Year,Month,Day,Hour,
name_day_of_week_processes(1)=day_of_week_processes(1)
title_day_of_week_processes(1)=day_of_week_processes(1)
cube_day_of_week_processes(1)=DZAuditCube
dimension_day_of_week_processes(1)=DZAuditDim
```

query_day_of_week_processes(1)=select NON EMPTY {[Measures].
[NORMAL_COUNT_S],[Measures].[ANOMALY_COUNT_S],[Measures].
[EXPIRED_COUNT_S],[Measures].[MISSING_COUNT_S],[Measures].
[GC_NOT_MET_COUNT_S],[Measures].[NORMAL_COUNT_A],[Measures].
[ANOMALY_COUNT_A],[Measures].[EXPIRED_COUNT_A],[Measures].
[MISSING_COUNT_A],[Measures].[GC_NOT_MET_COUNT_A]} ON columns, NON EMPTY [DZAuditDim.day_of_week_processes(1)].[All day_
of_week_processes(1)]} ON rows from [DZAuditCube]
hierarchy_day_of_week_processes(1)=(All),PROCESS_
NAME,PATH,EVENT_NAME,Year,Month,Day_of_Week,Hour,
name_day_of_week_processes(2)=day_of_week_processes(2)
title_day_of_week_processes(2)=day_of_week_processes(2)
cube_day_of_week_processes(2)=DZAuditCube
dimension_day_of_week_processes(2)=DZAuditDim
query_day_of_week_processes(2)=select NON EMPTY {[Measures].
[NORMAL_COUNT_S],[Measures].[ANOMALY_COUNT_S],[Measures].
[EXPIRED_COUNT_S],[Measures].[MISSING_COUNT_S],[Measures].
[GC_NOT_MET_COUNT_S],[Measures].[NORMAL_COUNT_A],[Measures].
[ANOMALY_COUNT_A],[Measures].[EXPIRED_COUNT_A],[Measures].
[MISSING_COUNT_A],[Measures].[GC_NOT_MET_COUNT_A]} ON columns, NON EMPTY {[DZAuditDim.day_of_week_processes(2)].[All day_
of_week_processes(2)]} ON rows from [DZAuditCube]
hierarchy_day_of_week_processes(2)=(All),PROCESS_NAME,EVENT_
NAME,Year,Month,Day_of_Week,Hour,

Appendix B

Schema definition file (dashboardMondrian.xml)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Schema name="DZAudit">
<Cube name="DZAuditCube">
<Table name="DZ_A_E_AGG" schema="PROCESS_AUDIT"/>
<Dimension name="DZAuditDim">
<Hierarchy allMemberName="All daily_processes(1)" hasAll="true"
name="daily_processes(1)">
<Level column="PROCESS_NAME" name="PROCESS_NAME"
uniqueMembers="true"/>
<Level column="PATH" name="PATH" uniqueMembers="true"/>
<Level column="EVENT_NAME" name="EVENT_NAME"
uniqueMembers="true"/>
<Level column="TS" name="Year" uniqueMembers="true">
```

```
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'YYYY')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Month" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'Mon')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Day" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'DD')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Hour" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'HH24')</SQL>
</KeyExpression>
</Level>
</Hierarchy>
<Hierarchy allMemberName="All daily_processes(2)" hasAll="true"
name="daily_processes(2)">
<Level column="PROCESS_NAME" name="PROCESS_NAME"
uniqueMembers="true"/>
<Level column="EVENT_NAME" name="EVENT_NAME"
uniqueMembers="true"/>
<Level column="TS" name="Year" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'YYYY')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Month" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'Mon')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Day" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'DD')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Hour" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'HH24')</SQL>
</KeyExpression>
```

```
</Level>
</Hierarchy>
<Hierarchy allMemberName="All day_of_week_processes(1)"
hasAll="true" name="day_of_week_processes(1)">
<Level column="PROCESS_NAME" name="PROCESS_NAME"
uniqueMembers="true"/>
<Level column="PATH" name="PATH" uniqueMembers="true"/>
<Level column="EVENT_NAME" name="EVENT_NAME"
uniqueMembers="true"/>
<Level column="TS" name="Year" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS, 'YYYY')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Month" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS, 'Mon')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Day_of_Week" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS, 'DY')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Hour" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS, 'HH24')</SQL>
</KeyExpression>
</Level>
</Hierarchy>
<Hierarchy allMemberName="All day_of_week_processes(2)"
hasAll="true" name="day_of_week_processes(2)">
<Level column="PROCESS_NAME" name="PROCESS_NAME"
uniqueMembers="true"/>
<Level column="EVENT_NAME" name="EVENT_NAME"
uniqueMembers="true"/>
<Level column="TS" name="Year" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS, 'YYYY')</SQL>
</KeyExpression>
</Level>
```

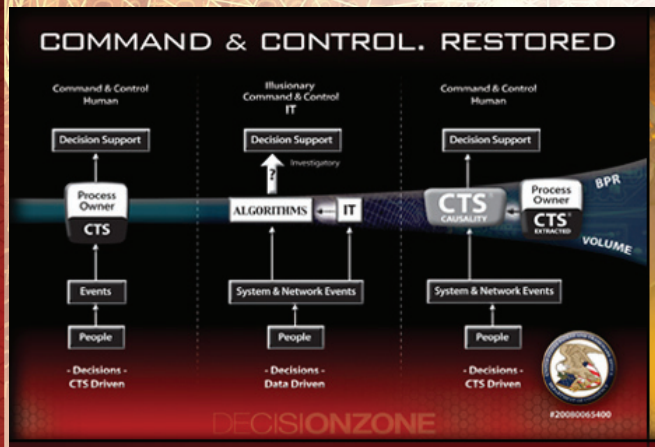
```
<Level column="TS" name="Month" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'Mon')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Day_of_Week" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'DY')</SQL>
</KeyExpression>
</Level>
<Level column="TS" name="Hour" uniqueMembers="true">
<KeyExpression>
<SQL dialect="oracle">TO_CHAR(TS,'HH24')</SQL>
</KeyExpression>
</Level>
</Hierarchy>
</Dimension>
<Measure aggregator="sum" column="NORMAL_COUNT" format-
String="#.0" name="NORMAL_COUNT_S" visible="true"/>
<Measure aggregator="avg" column="NORMAL_COUNT" format-
String="#.0" name="NORMAL_COUNT_A" visible="true"/>
<Measure aggregator="sum" column="ANOMALY_COUNT" format-
String="#.0" name="ANOMALY_COUNT_S" visible="true"/>
<Measure aggregator="avg" column="ANOMALY_COUNT" format-
String="#.0" name="ANOMALY_COUNT_A" visible="true"/>
<Measure aggregator="sum" column="EXPIRED_COUNT" format-
String="#.0" name="EXPIRED_COUNT_S" visible="true"/>
<Measure aggregator="avg" column="EXPIRED_COUNT" format-
String="#.0" name="EXPIRED_COUNT_A" visible="true"/>
<Measure aggregator="sum" column="MISSING_COUNT" format-
String="#.0" name="MISSING_COUNT_S" visible="true"/>
<Measure aggregator="avg" column="MISSING_COUNT" format-
String="#.0" name="MISSING_COUNT_A" visible="true"/>
<Measure aggregator="sum" column="GC_NOT_MET_COUNT" format-
String="#.0" name="GC_NOT_MET_COUNT_S" visible="true"/>
<Measure aggregator="avg" column="GC_NOT_MET_COUNT" format-
String="#.0" name="GC_NOT_MET_COUNT_A" visible="true"/>
</Cube>
</Schema>
```

Appendix C

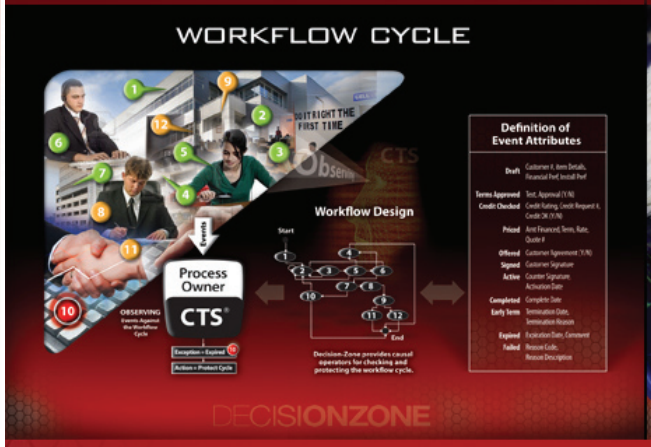
Bridge configuration file schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="bridgeConfig">
<xs:annotation>
<xs:documentation>Comment describing your root element</
xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="listener">
<xs:complexType>
<xs:attribute name="topic" type="xs:string" use="required"/>
<xs:attribute name="provider" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="dbConnection">
<xs:complexType>
<xs:attribute name="jdbcDriver" type="xs:string" use="required"/>
<xs:attribute name="connectionURL" type="xs:string" use="required"/>
<xs:attribute name="user" type="xs:string" use="required"/>
<xs:attribute name="password" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="eventMapping" minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:complexContent>
<xs:extension base="EventMappingType"/>
</xs:complexContent>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

```
<xs:complexType name="EventMappingType">
<xs:annotation>
<xs:documentation>event mapping type</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="fieldMapping" maxOccurs="unbounded">
<xs:complexType>
<xs:complexContent>
<xs:extension base="FieldMappingType"/>
</xs:complexContent>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="tableName" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="FieldMappingType">
<xs:annotation>
<xs:documentation>field mapping type</xs:documentation>
</xs:annotation>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="eventFieldType" use="required">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="java.lang.String"/>
<xs:enumeration value="java.lang.Number"/>
<xs:enumeration value="java.lang.Boolean"/>
<xs:enumeration value="java.util.Date"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="tableFieldName" type="xs:string" use="required"/>
<xs:attribute name="tableFieldType" type="xs:string" use="required"/>
<xs:attribute name="tableFieldLength" type="xs:string"
use="optional"/>
<xs:attribute name="dateFormat" type="xs:string" use="optional"/>
</xs:complexType>
</xs:schema>
```



CYBER-PROTECTION



DECISIONZONE

CYBER-PROTECTION

A SINGLE ANOMALOUS EVENT CAN CAUSE CATASTROPHIC CRISIS.

EACH INTERNAL OR EXTERNAL EVENT MUST BE CHECKED FOR ANOMALIES.

ANOMALIES OCCUR WHEN AN EVENT VIOLATES THE
PRE-DEFINED EXECUTION PATTERN CYCLE.

AN EXECUTION PATTERN CYCLE IS A CAUSAL EVENT GRAPH BASED ON CTS.

CURRENT SOFTWARE LANGUAGES CAN ONLY CHECK FOR ANOMALOUS EVENTS
AFTER THE COMPLETION OF THE EXECUTION PATTERN CYCLE OR WHEN
THE CATASTROPHIC CRISIS HAS ALREADY HAPPENED.

DECISION-ZONE HAS DEVELOPED A CAUSAL BASED SOFTWARE LANGUAGE FOR BUILDING A COMPREHENSIVE SYSTEM THAT CAN IN REAL TIME CHECK A SINGLE EVENT AGAINST ALL POSSIBLE ANOMALIES USING THE CAUSAL GRAPH (CTS) DURING THE PATTERN EXECUTION CYCLE TO PROTECT AGAINST ANY CATASTROPHIC CRISIS.

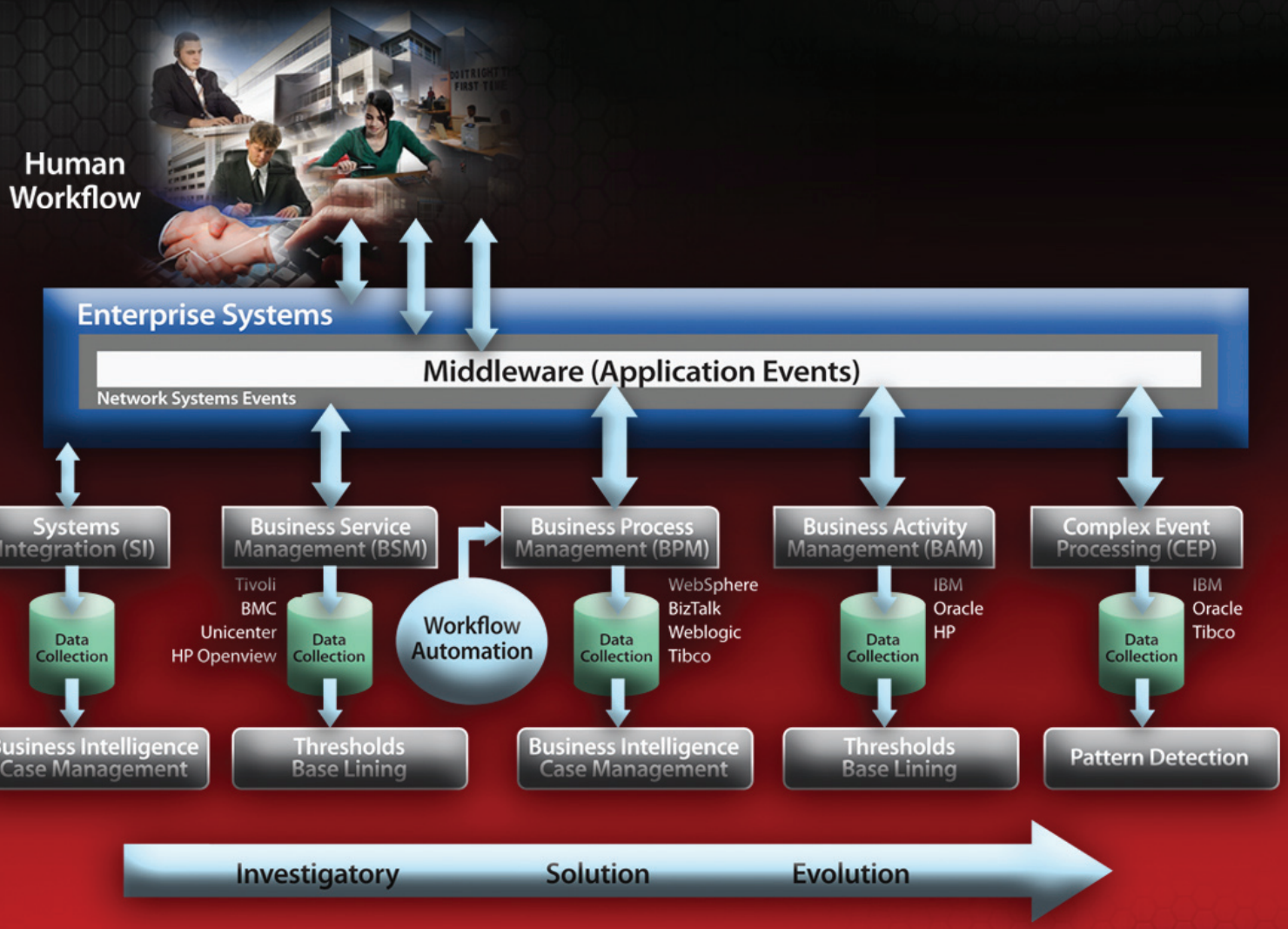
DECISION-ZONE'S DZAUDIT SYSTEM OUTPUTS CONSOLIDATED AUDIT TRAILS FOR BOTH ANOMALOUS AND NORMAL EVENTS FOR FUTURE BUSINESS INTELLIGENCE ANALYTICS.

DECISION-ZONE HAS ADDITIONALLY BUILT A PATTERN CYCLE DISCOVERY SYSTEM FOR CHECKING THE VALIDITY OF THE DEFINED PATTERN CYCLES.

DECISIONZONE

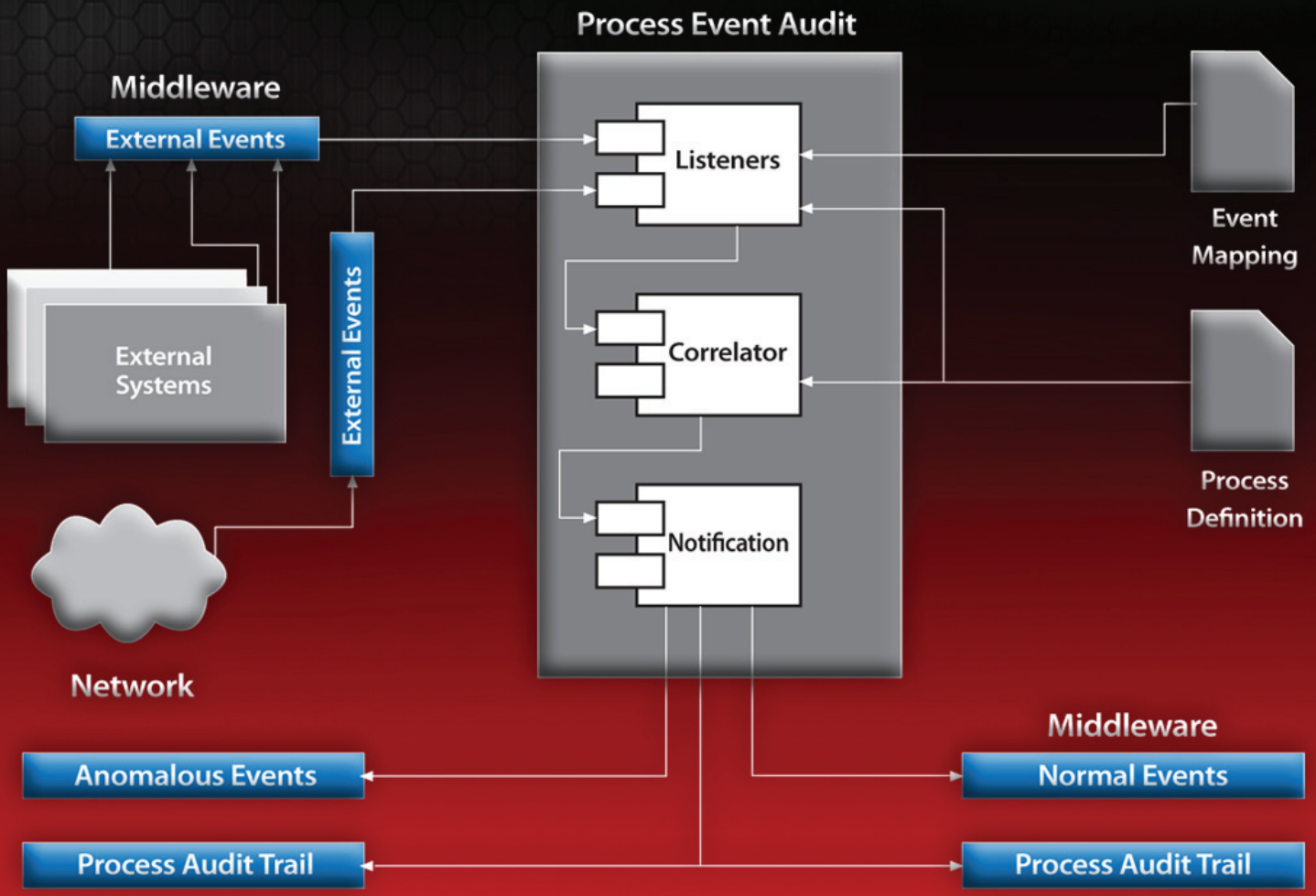
HOW PROCESS OWNER PROTECTS HIS BUSINESS WORKFLOW

CURRENT SOLUTIONS CAN ONLY INVESTIGATE FRAUD. THE COST OF INVESTIGATING 1 TRILLION DOLLARS OF FRAUD IS 2.8 TRILLION DOLLARS

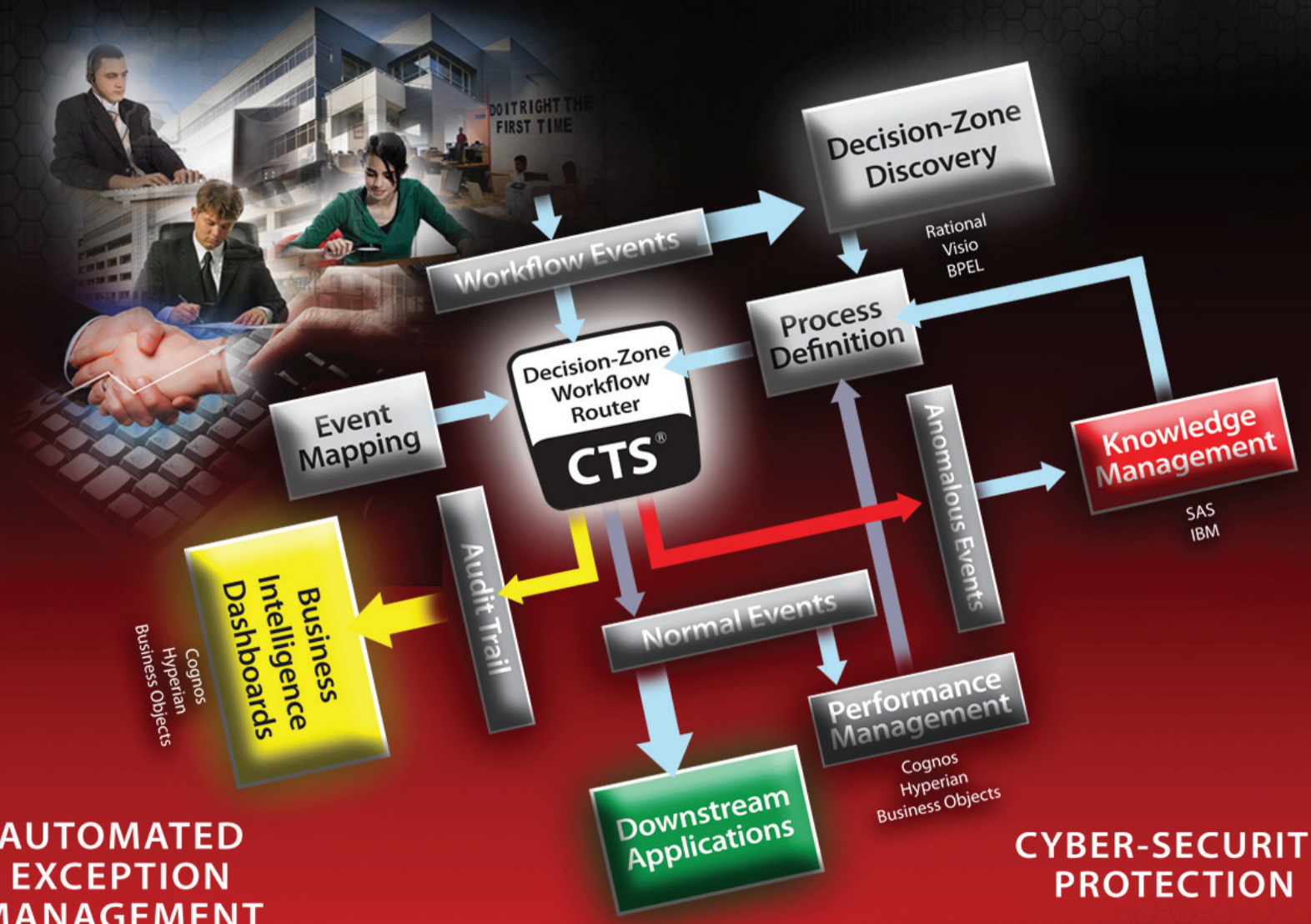


DECISIONZONE

DECISION-ZONE'S PATENTED CYBER-PROTECTION ARCHITECTURE



THE DEPLOYMENT: DECISION-ZONE'S CYBER-PROTECTION SOLUTION



Appendix E

Case Study 1 - Employee Provisioning Historical

Preface

This sample package demonstrates the basic functionality of the DZ Audit. It walks the user through an end to end typical implementation scenario for a real-time process audit.

It is assumed that the user has read the 'DZAuditManual' document and has all required concepts and training if a process definition file needs to be edited or created (UML statechart diagrams).

How this Guide is organized

This document contains the following sections:

Chapter 1, "Introduction"

Chapter 2, "Configuration"

Chapter 3, "Dependencies"

Chapter 4, "Running the sample"

Conventions

This manual uses the following conventions:

- Monospace
  - Specifies file names, object names, and programming code.

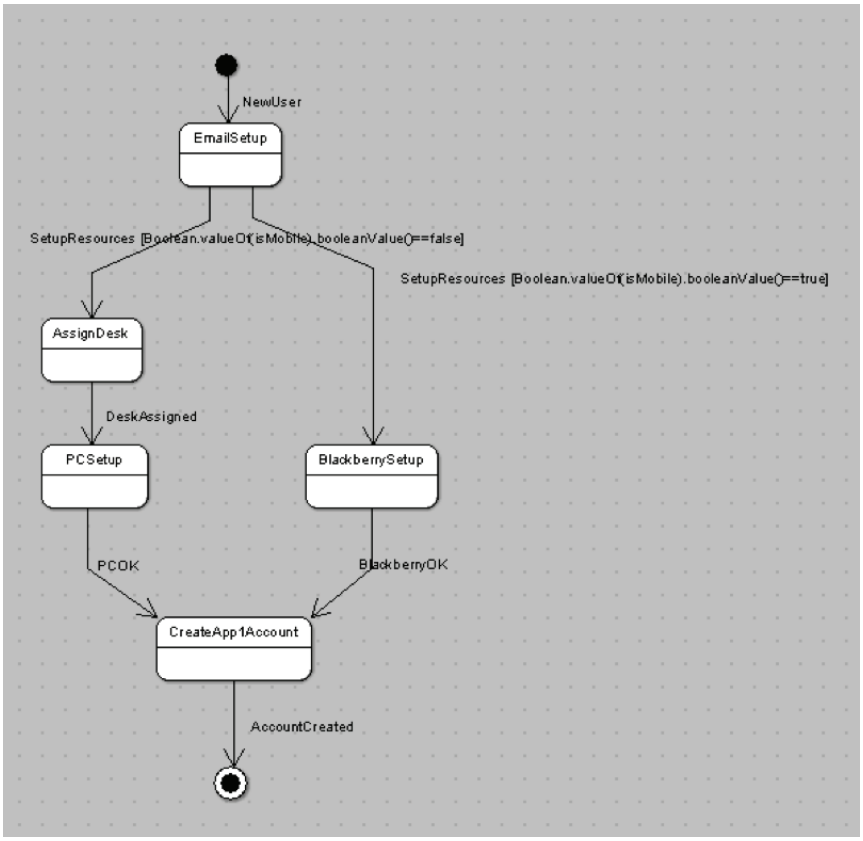
- Italics
  - Identifies a variable
  - Indicates a value that you must enter
  - Introduces new terminology, highlights and provides emphasis

- Bold
  - Highlights items and indicates specific items in a graphical user interface.

Introduction

Overview

The sample will perform a process audit of an imaginary employee provisioning process that activates a new user.



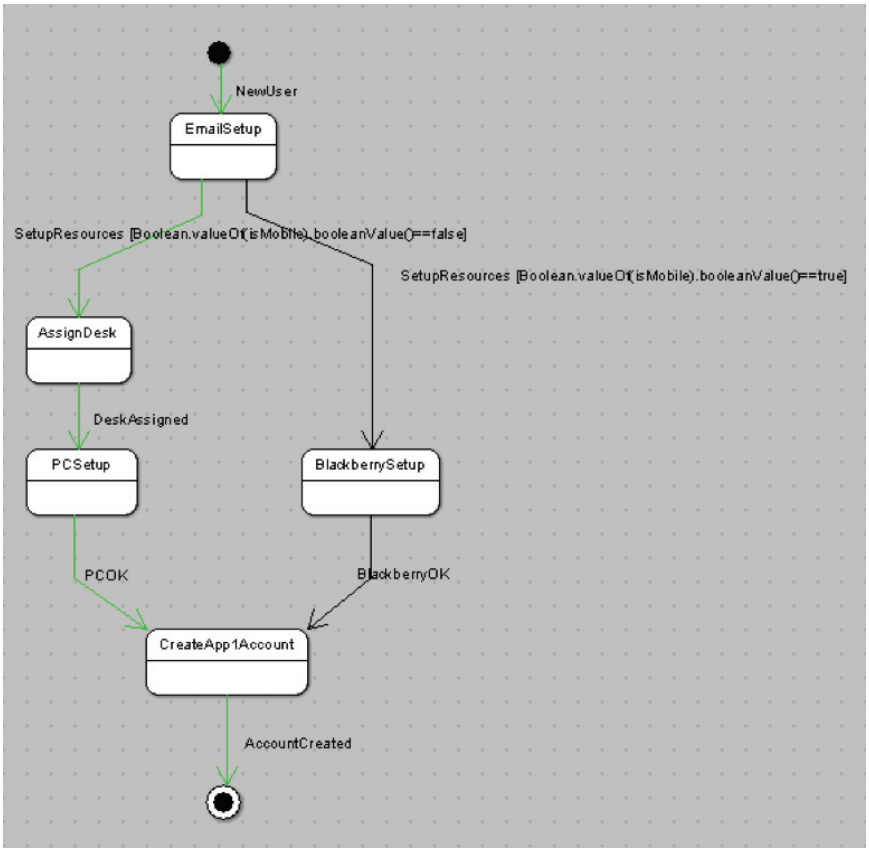
The process assumes that two types of users are configured and both types use 'App1' as their primary application. To gain access to the application, an account will have to be created.

For the 'Office' type, a desk needs to be assigned after which a computer has to be configured with the appropriate OS and basic applications.

For the 'Mobile' type, a communication device (ex. Blackberry) with its remote access tools needs to be setup.

Activation of a 'Office' user

The green path depicts the sequence of events pushed when an 'Office' type user is added to the system.

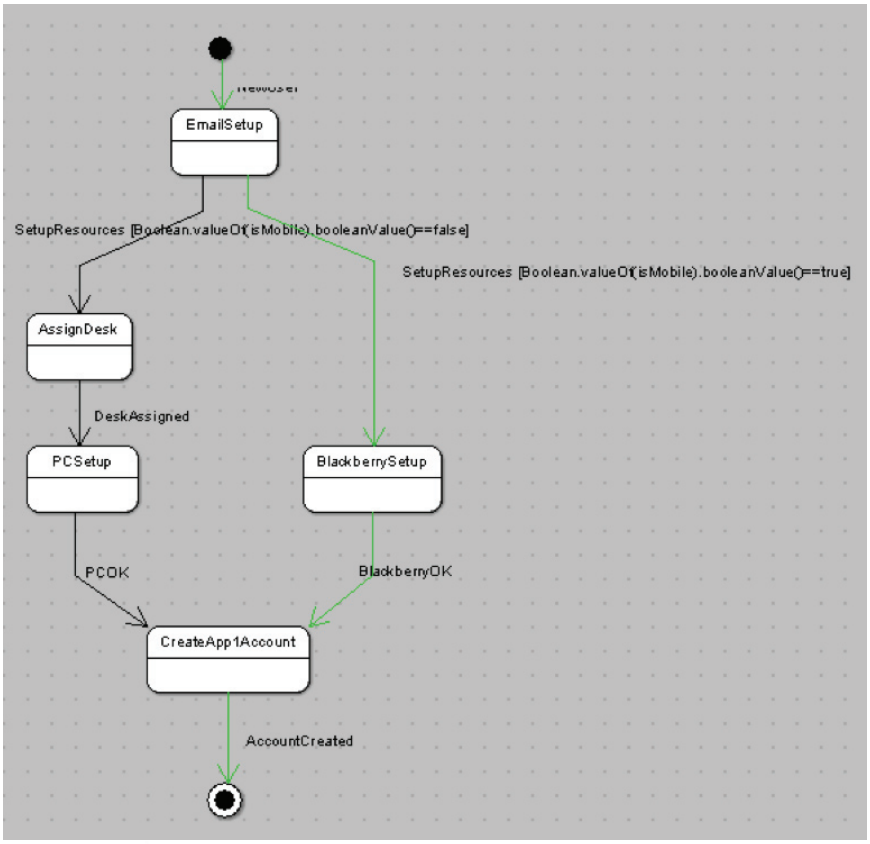


The following events are pushed:

– 'NewUser – startup event
– SetupResources – the Boolean field 'isMobile' is false.
– DeskAssigned – event pushed after a desk was assigned
– PCOK – event pushed after the corresponding PC was properly setup and installed.
– AccountCreated – event pushed after the account in the 'App1' was created.

Activation of a 'Mobile' user

The green path depicts the sequence of events pushed when a 'Mobile type user is added to the system.

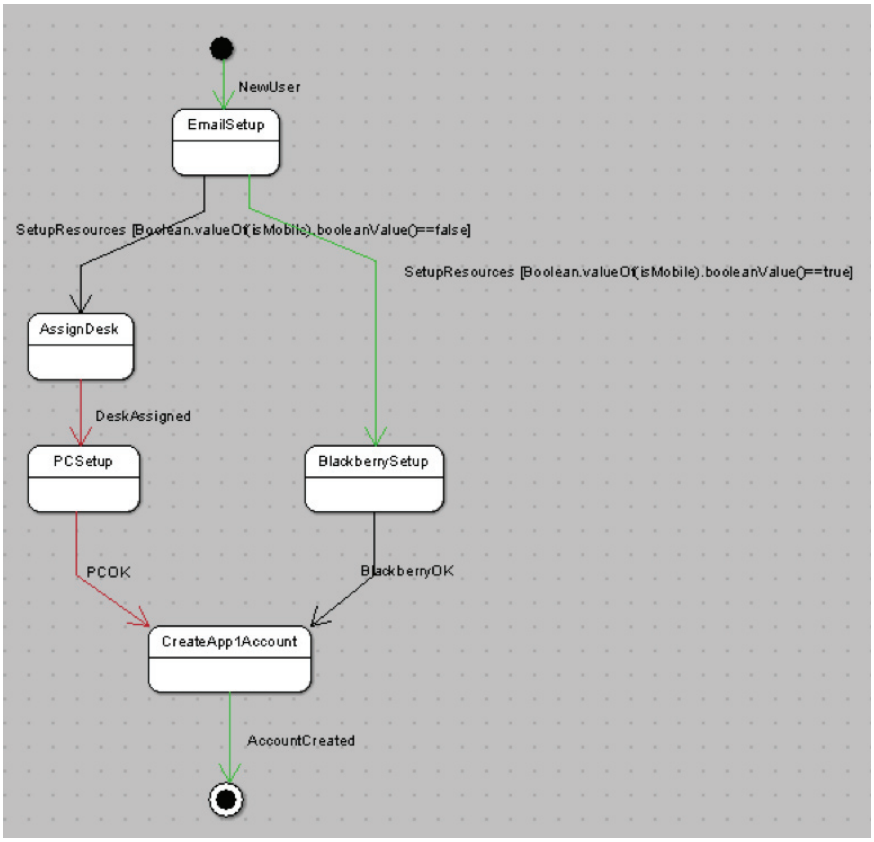


The following events are pushed:

– 'NewUser – startup event
– SetupResources – the Boolean field 'isMobile' is true.
– BlackBerryOK – event pushed after the comm. Device was setup.
– AccountCreated – event pushed after the account in the 'App1' was created.

Invalid activation sequence

The green path depicts the sequence of events pushed when a 'Mobile type user is added to the system but the activities associated wit an 'Office' type user are performed.



The following events are pushed:

- 'NewUser – startup event (GREEN)
- SetupResources – the boolean field 'isMobile' is true. (GREEN)
- DeskAssigned – event pushed after a desk was assigned (RED)
- PCOK – event pushed after the corresponding PC was properly setup and installed. (RED)
- AccountCreated – event pushed after the account in the 'App1' was created. (GREEN)

Configuration

Sample files

EmployeeProvisioning.zargo – UML process definition file that is used as a process definition source. To view the process statechart diagram, select 'Edit Process Definition' from the context menu of 'Process definition' node from the descriptor.

After the editor is active, select 'Diagram-centric' option from the 'Perspective menu'. Drill down to 'UserResourceInfoStatechartDiagram' and double-click it.

EmployeeProvisioning.xmi – XMI file exported from EmployeeProvisioning.zargo. This file will be used in the mapping process.

NewEmployeeProvisioning.pad – Descriptor for this sample. It contains references to all resources in this sample.

NewEmployeeProvisioning.pam – Mapping for this sample.

EmployeeProvisioning.pas – Server configuration for the runtime phase of the sample

EmployeeProvisioning.cslt - Causality capture file. Sample data to be used for data exploration/modeling.

processAuditServer.bat – Batch file to startup DZAudit with the appropriate parameters for this sample.

run_HSQLDB_Server.bat – Batch file that will run the HSQLDB server.

The HSQLDB server store all the historical data.

run_HSQLDB_ManagerSwing.bat – Batch file that will run the HSQLDB Manager. HSQLDB Manager is a swing application designed to manage the HSQLDB database and execute SQL statements.

The sample descriptor includes a business event definition in a form of a pattern called 'Office Setup'. It will match a sequence of events like 'DeskAssigned' followed by a 'PCOK'. For this sample, the pattern match will both 'NewOfficeUser' and 'NewBadUser' scenarios.

Note: *To listen for the pattern match business event, select 'Listen' from the context menu of NewEmployeeProvisioning.pad/BusinessEvents/Pattern 'Office Setup' node*

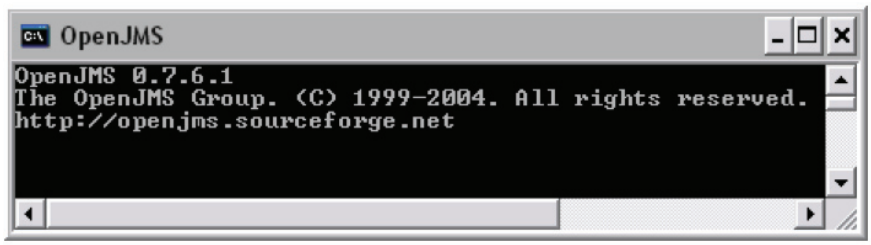
Dependencies

Middleware

The sample uses 'OpenJMS' as the default middleware layer to publish and subscribe to the events of interest. Any providers in any combination can be used to retrieve the events of interest for the audit process.

Startup

The middleware layers need to be active before the DZAudit engine is launched. Go to 'samples\openjms-0.7.6.1\bin' directory and select 'startup.bat' batch file to startup OpenJMS. After startup, the following window will be visible:



Shutdown

After the sample walkthrough is completed, the middleware layer needs to be shutdown. Go to 'samples\openjms-0.7.6.1\bin' directory and select 'shutdown.bat'

Database

The sample use a HSQLDB database to store all the historical data.

Startup

The HSQLDB database need to be active before the DZAudit engine is launched. To startup the database execute 'run_HSQLDB_Server.bat'.

Shutdown

After the sample walkthrough is completed, the database needs to be shutdown. There are two ways to shutdown the database:

- connect to HSQLDB and execute SHUTDOWN SQL
- run HSQLDB Manager application by executing 'run_HSQLDB_ ManagerSwing.bat'
- connect to database server using the following parametrs:
- Type: HSQL Database Engine Server
- Driver: org.hsqldb.jdbcDriver
- URL: jdbc:hsqldb:hsql://localhost/EmployeeProvisioning
- User: EMPLOYEE_PROVISIONING
- Password: EMPLOYEE_PROVISIONING
- Execute command: SHUTDOWN
- From command line, use [Ctrl]+[C] to abort abruptly

Running the sample

- Startup the middleware layer.
- Startup the database.
- Startup the DZAudit server by selecting 'processAuditServer.bat' from 'samples\EmployeeProvisioning' directory.
- Check the console output for error or warning messages.
- Startup IDE
- Start listening for RawEvents by selecting 'Listen' from the 'New EmployeeProvisioning.pad/BusinessEvents/RawEvents' node.
- Mount the DZ audit server by selecting 'Mount/DZAudit server' from the 'Filesystems' node, and replace [hostname] with localhost and [port] with 5099
- Start the batch processing by selecting 'User action/Start batch' from the 'DZAudit server/NewEmployeeProvisioning.pad' or by executing 'startBatch.bat'.
- Shutdown server.
- Shutdown the middleware layer

Appendix F

Case Study 2 - Employee Provisioning Real-Time

Preface

This sample package demonstrates the basic functionality of the DZ Audit. It walks the user through an end to end typical implementation scenario for a real-time process audit.

It is assumed that the user has read the 'DZAuditManual' document and has all required concepts and training if a process definition file needs to be edited or created (UML statechart diagrams).

How this Guide is organized

This document contains the following sections:

Chapter 1, "Introduction"

Chapter 2, "Configuration"

Chapter 3, "Dependencies"

Chapter 4, "Running the sample"

Conventions

This manual uses the following conventions:

- Monospace
  - Specifies file names, object names, and programming code.

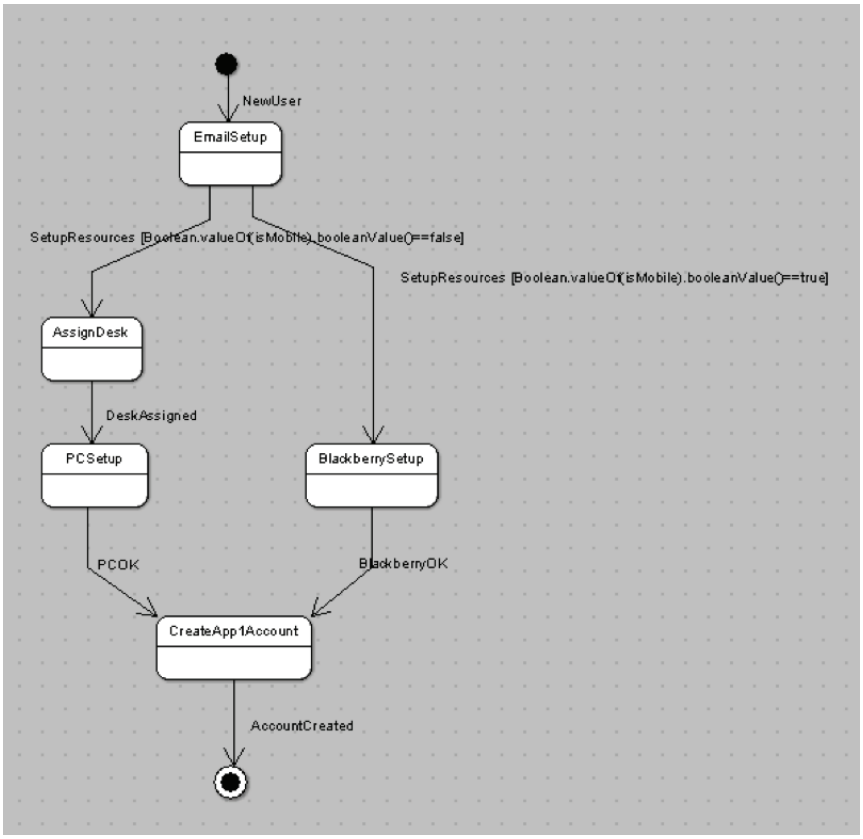
- Italics
  - Identifies a variable
  - Indicates a value that you must enter
  - Introduces new terminology, highlights and provides emphasis

- Bold
  - Highlights items and indicates specific items in a graphical user interface.

Introduction

Overview

The sample will perform a process audit of an imaginary employee provisioning process that activates a new user.



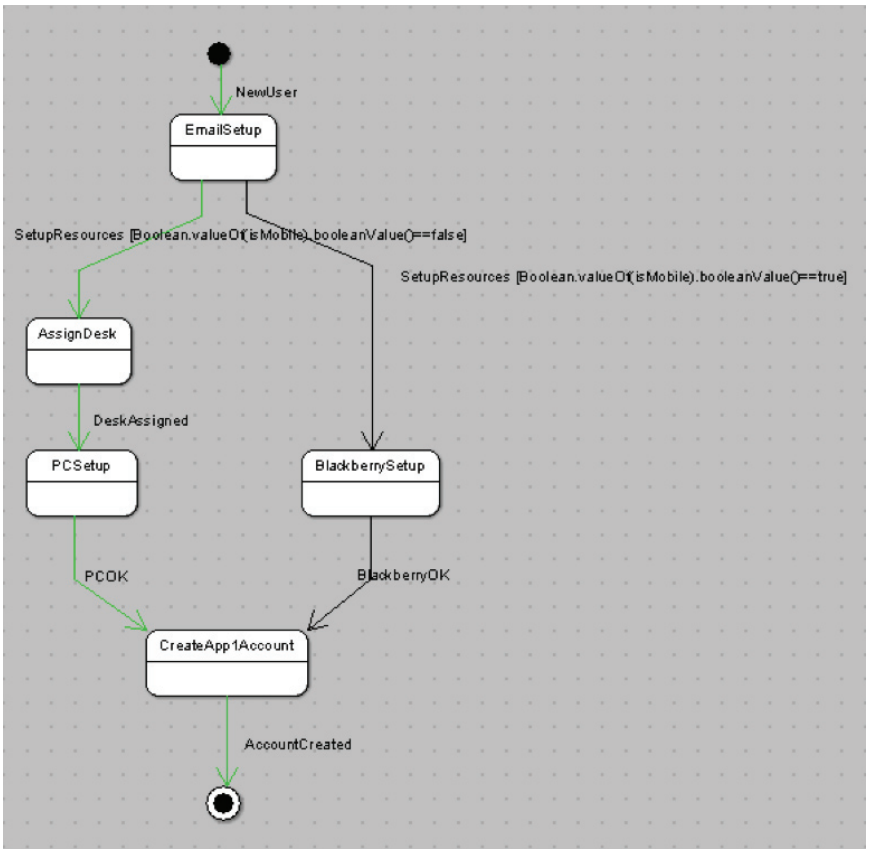
The process assumes that two types of users are configured and both types use 'App1' as their primary application. To gain access to the application, an account will have to be created.

For the 'Office' type, a desk needs to be assigned after which a computer has to be configured with the appropriate OS and basic applications.

For the 'Mobile' type, a communication device (ex. Blackberry) with its remote access tools needs to be setup.

Activation of a 'Office' user

The green path depicts the sequence of events pushed when an 'Office' type user is added to the system.

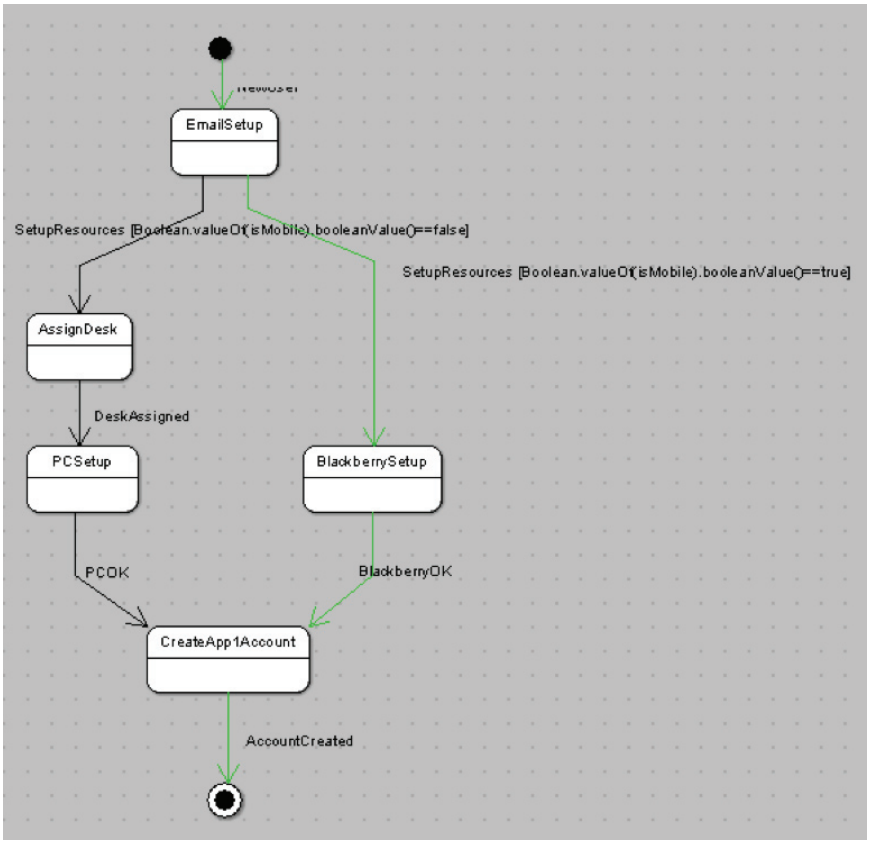


The following events are pushed:

– 'NewUser – startup event
– SetupResources – the Boolean field 'isMobile' is false.
– DeskAssigned – event pushed after a desk was assigned
– PCOK – event pushed after the corresponding PC was properly setup and installed.
– AccountCreated – event pushed after the account in the 'App1' was created.

Activation of a 'Mobile' user

The green path depicts the sequence of events pushed when a 'Mobile type user is added to the system.

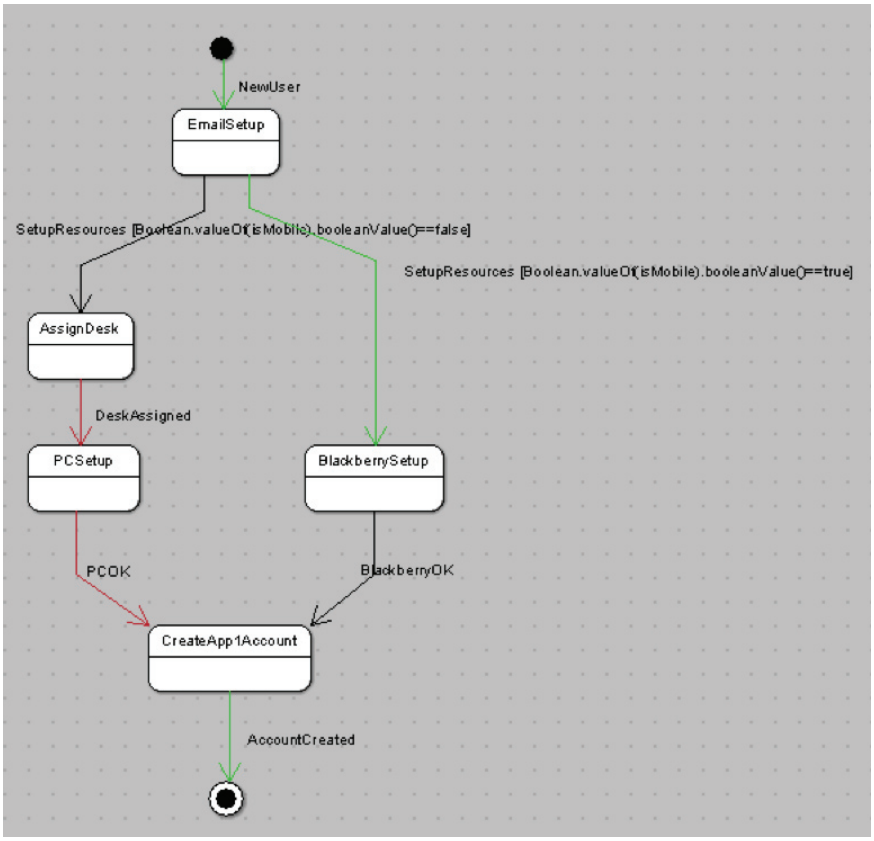


The following events are pushed:

– 'NewUser – startup event
– SetupResources – the Boolean field 'isMobile' is true.
– BlackBerryOK – event pushed after the comm. Device was setup.
– AccountCreated – event pushed after the account in the 'App1' was created.

Invalid activation sequence

The green path depicts the sequence of events pushed when a 'Mobile type user is added to the system but the activities associated wit an 'Office' type user are performed.



The following events are pushed:

– 'NewUser – startup event (GREEN)
– SetupResources – the boolean field 'isMobile' is true. (GREEN)
– DeskAssigned – event pushed after a desk was assigned (RED)
– PCOK – event pushed after the corresponding PC was properly setup and installed. (RED)
– AccountCreated – event pushed after the account in the 'App1' was created. (GREEN)

Configuration

Sample files

EmployeeProvisioning.zargo – UML process definition file that is used as a process definition source. To view the process statechart diagram, select 'Edit Process Definition' from the context menu of 'Process definition' node from the descriptor.

After the editor is active, select 'Diagram-centric' option from the 'Perspective menu'. Drill down to 'UserResourceInfoStatechartDiagram' and double-click it.

EmployeeProvisioning.xmi – XMI file exported from EmployeeProvisioning.zargo. This file will be used in the mapping process.

NewEmployeeProvisioning.pad – Descriptor for this sample. It contains references to all resources in this sample.

NewEmployeeProvisioning.pam – Mapping for this sample.

EmployeeProvisioning.pas – Server configuration for the runtime phase of the sample

EmployeeProvisioning.cslt - Causality capture file. Sample data to be used for data exploration/modeling.

processAuditServer.bat – Batch file to startup DZAudit with the appropriate parameters for this sample.

NewOfficeUser.bat –Batch file that will push events that match the addition of a new 'Office' type employee.

NewMobileUser.bat - Batch file that will push events that match the addition of a new 'Mobile' type employee.

NewBadUser.bat – Batch file that will push events that do not comply with the process definition. It will push the events associated with a 'Office ' type employee but the event associated with the resource allocation will be consistent to a 'Mobile' type user.

The sample descriptor includes a business event definition in a form of a pattern called 'Office Setup'. It will match a sequence of events like 'DeskAssigned' followed by a 'PCOK'. For this sample, the pattern match will both 'NewOfficeUser' and 'NewBadUser' scenarios.

Note: *To listen for the pattern match business event, select 'Listen' from the context menu of NewEmployeeProvisioning.pad/BusinessEvents/Pattern 'Office Setup' node*

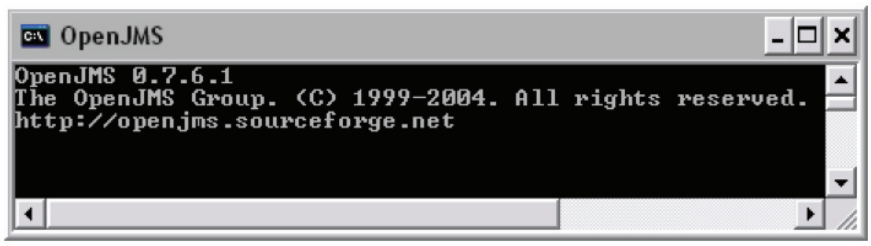
Dependencies

Middleware

The sample uses 'OpenJMS' as the default middleware layer to publish and subscribe to the events of interest. Any providers in any combination can be used to retrieve the events of interest for the audit process.

Startup

The middleware layers need to be active before the DZAudit engine is launched. Go to 'samples\openjms-0.7.6.1\bin' directory and select 'startup.bat' batch file to startup OpenJMS. After startup, the following window will be visible:



Shutdown

After the sample walkthrough is completed, the middleware layer needs to be shutdown. Go to 'samples\openjms-0.7.6.1\bin' directory and select 'shutdown.bat'

Database

The sample use a HSQLDB database to store all the historical data.

Startup

The HSQLDB database need to be active before the DZAudit engine is launched. To startup the database execute 'run_HSQLDB_Server.bat'.

Shutdown

After the sample walkthrough is completed, the middleware layer needs to be shutdown. Go to 'samples\openjms-0.7.6.1\bin' directory and select 'shutdown.bat'

Running the sample

- Startup the middleware layer.
- Startup the database.
- Startup the DZAudit server by selecting 'processAuditServer.bat' from 'samples\EmployeeProvisioning' directory.
- Check the console output for error or warning messages.
- Startup IDE
- Start listening for RawEvents by selecting 'Listen' from the 'New EmployeeProvisioning.pad/BusinessEvents/RawEvents' node.
- Push desired event sets by selecting the appropriate batch file (NewOfficeUser/NewMobileUser/NewBadUser)
  - The detected event sets (normal or anomalies) will now be present in the inspection panel of the listener.
  - Repeat this step as required
- Shutdown server.
- Shutdown the middleware layer

Appendix G

Case Study 3 - Manual Workflow

Preface

How this Guide is organized

This document contains the following sections:

Chapter 1, "Overview"

Chapter 2, "Functional Description"

Conventions

This manual uses the following conventions:

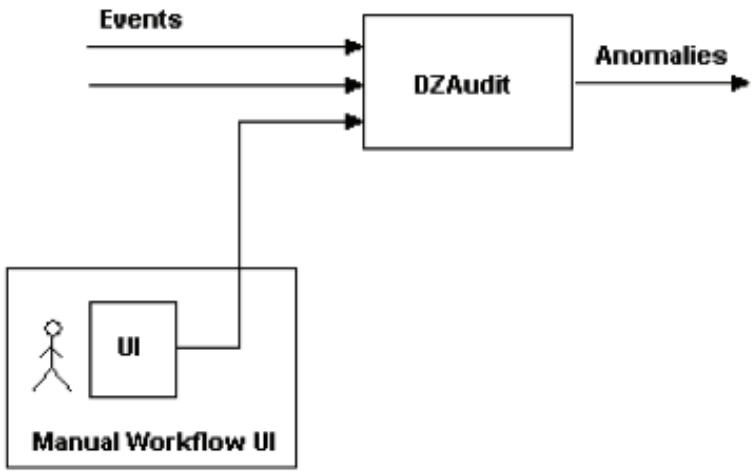
- Monospace
  - Specifies file names, object names, and programming code.

- Italics
  - Identifies a variable
  - Indicates a value that you must enter
  - Introduces new terminology, highlights and provides emphasis

- Bold
  - Highlights items and indicates specific items in a graphical user interface.

Overview

The user interface of DZAudit Manual Workflow provides the means for creating events to be sent to DZAudit engine, based on manual user inputs. DZAudit will capture these events, process them and detect eventual anomalies.



Functional Description

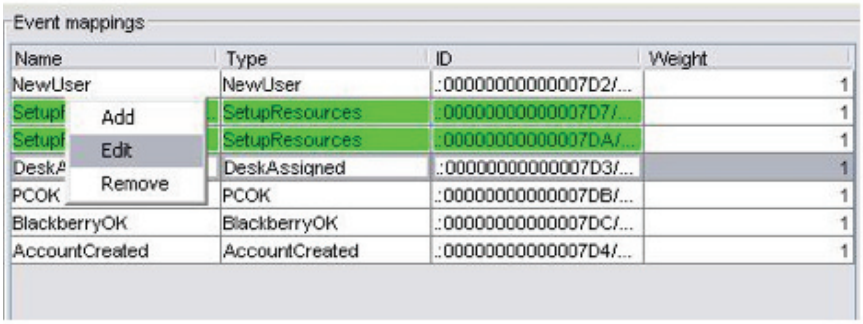
Mapping 'Manual Workflow' provider events

To enable the usage of the manual workflow provider in the IDE, the lookup file ('lookup.xml') should include the following information:

```
<provider name="ManualWorkflow" type="in">
        <class>com.decisionzone.businessbroker.event.
manualWorkflow.ManualWorkflowSignalFactory </class>
        <parameters>
                <parameter name="TransportProvider" type="in">
                        <className>java.lang.String</className>
                        <defaultValue>TopicJMS</defaultValue>
                </parameter>
                <parameter name="Role" type="in">
                        <className>java.lang.String</className>
                        <defaultValue></defaultValue>
                </parameter>
        </parameters>
</provider>
```

In the DZAudit IDE, to map events with 'Manual Workflow' provider, select an Event Mapping node, right click and select **'Edit Mapping'**.

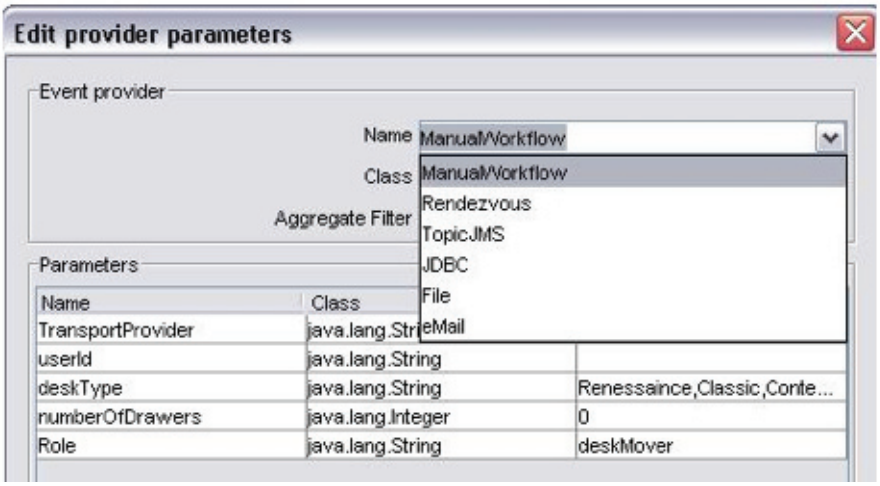
In the 'Event mappings' panel, select desired event, right click on it and select **'Edit'.**



Click **'Event Provider'** button. A window for the event provider parameters will open.

In the 'Event Provider' section, choose 'ManualWorkflow' as a provider, from the 'Name' drop-down list.

In 'Parameter' section, add parameters for the provider, specifying the following fields:
- 'Name' - the name of the parameter.
- 'Class' - parameter class (i.e. java.lang.String, java.lang.Integer).
- 'Default' - represent the default values, if any. For more than one default values, separate their values with a comma.



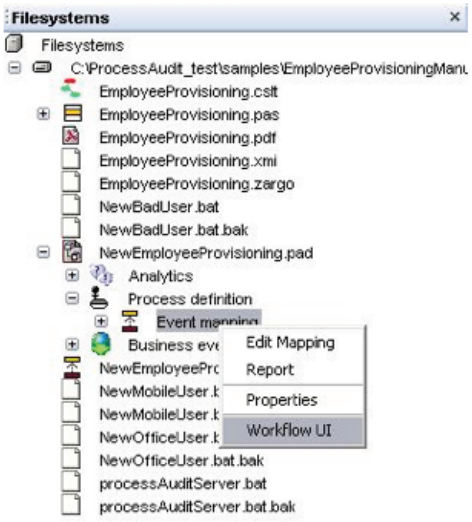
The parameters that have been specified in the 'lookup.xml' file should also be configured here.
Once the provider's parameters have been configured, click 'Close' then 'Save', to persist the changes.

NOTE: *If no default value has been specified for 'Role' parameter in the mapping file, all users configured to have access to the 'Manual Workflow' web application will be able to provide the event data.*

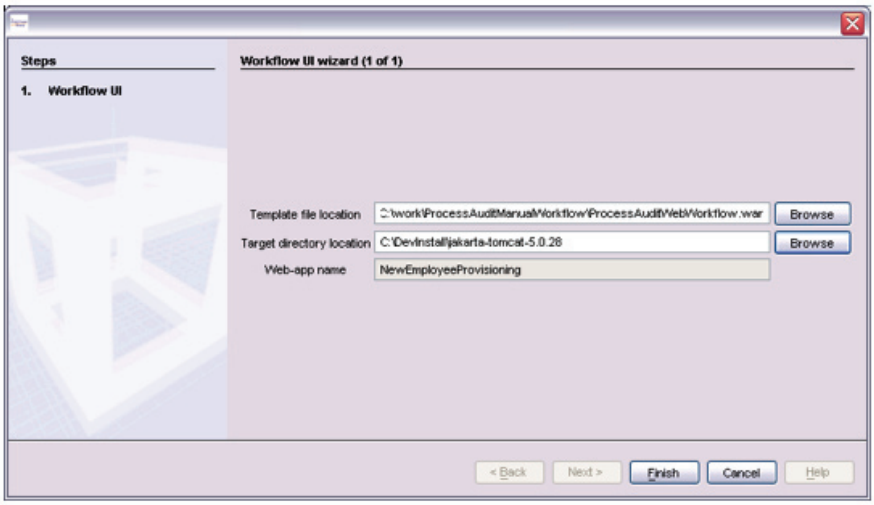
If a default value is specified for this parameter, only users that have that role assigned will have access to input the event's data.

Deploying 'Manual Workflow' UI

Select an Event Mapping node, right click and select **'Workflow UI'.**
A wizard window will open, pointing the user to provide the following information:



- Template file location – use Browse button to browse through the file directory and select a .war template file for later deployment.
- Target directory location – use Browse button to select a target directory where the .war file will be copied and deployed.



'Web-app name' field represents the name of the web application and is determined by the selected node's mapping file name.

NOTE: *After deployment, whenever the mapping file (.pam) changes, just overwrite the old mapping file with the new one in the /WEB-INF directory of the web application.*

'Manual Workflow' User Interface

To access the application through a web browser, use the following address:

http://<host_ip_address>:<port>/<webapp_name>/<webapp_name>
where:
<host_ip_address> - IP address of the host.
<port> - the server's port.
<webapp_name> - web application name (see Workflow UI wizard window), corresponding to the selected event's mapping name.
User authentication is required, so please provide a user name and a password.

Creating events

The index page will display only those events that in the mapping file have been configure with 'ManualWorkflow' provider and are visible for the role assigned to the user.

Select one using the corresponding radio button, then click '**Continue'.**

Next page will list the provider's parameters (see mapping events in 'Event provider parameters' window) and the user has to fill in their values. If a default value has been specified for the parameter that default value will be shown. If the parameter's default value is represented by a comma separated list, it will be translated into a drop-down list.

Click **'Submit'** for form submission. Upon submission an event will be created and sent and also an entry will be added to the history file (see 'View History' paragraph).

A confirmation page will be displayed upon submission.

'Go to index page' will redirect to the first (index) page.
'View History' will display history entries.

'Manual Workflow' can be used for automatic posting of event data by third party application by invoking an URL post with the following

parameters:
1. <input type="hidden" name="eventName" value="<event-name>" />
(<event-name> represents the event name that is to be created and sent, as declared in the mapping)

2. <input type="hidden" name="Role" value="<role_name>" />
(<role_name> represents the role name declared in the mapping file for this event. Leave blank if no role has been specified.)

3. <input type="hidden" name="<parameter_name>_className" value="<className>" />
<input type="text" name="<parameter_name>_value" value="<parameter_value>" size="30"/>

For each of the event's parameters defined in the mapping file with name starting with "ui_", except the default ones 'Role' and 'TopicPublisher', the URL post must contain the two lines mentioned above, where:

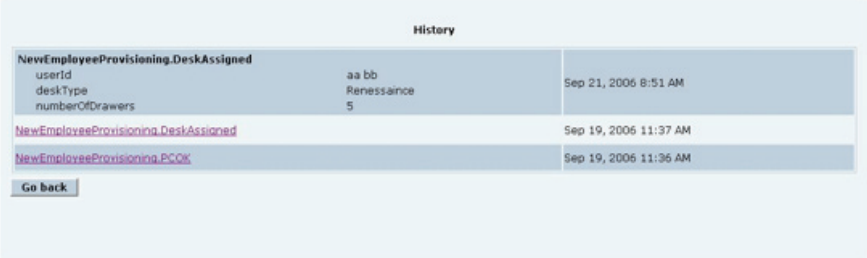
<parameter_name> represents the parameter name.
<className>" represents the parameter's class name, i.e. "java.lang.String", "java.lang.Integer".
<parameter_value> represents the parameter value.

Example for <parameter_name> = userId
<className> = java.lang.String
<parameter_value> = john001
<input type="hidden" name=" userId_className" value="java.lang.String"/>
<input type="text" name=" userId_value" value="john001" size="30"/>

View History

Every time an event is created and sent, an entry will be added to the history file.

To view the content of each history entry, click on its link. Information displayed refers to the event's field names and values, along with creation date and time.



NOTE: *A maximum history entries should be defined in the web application descriptor file (web.xml), as 'maxHistoryEntries' parameter. If not, a default value of 100 is set.*

Appendix H: Computations and Theory of Patterns

1.1 Introduction

This chapter introduces the basic concepts and operations of RAPIDE computations. Then the RAPIDE pattern language is defined in terms of these operations.

1.2 Computations

Computations consist of *events*, which are uniquely identifiable tuples of values. The Event() type is defined in the Predefined Types LRM.

The *identity* relation $==$ is a congruence relation; that is, it satisfies the *equivalence axioms*:

$$\begin{array}{lll}
(\forall \text{ event } e)e == e & & \text{(reflexivity)} \\
(\forall \text{ event } e_1, e_2)e_1 == e_2 & \rightarrow \quad e_2 == e_1 & \text{(symmetry)} \\
(\forall \text{ event } e_1, e_2, e_3)(e_1 == e_2 \wedge e_2 == e_3) & \rightarrow \quad e_1 == e_3 & \text{(transitivity)}
\end{array}$$

as well as the *functional substitutivity axiom schema*, for every n-ary function f and every i from 1 to n:

$$\begin{array}{l}
(\forall \text{ event } e_1, e_2)(\forall z_1, z_2, ...z_n)e_1 == e_2 \quad \rightarrow \\
\quad f(z_1, ..., z_i, e_1, z_i + 1, ..., z_n) == f(z_1, ..., z_i, e_2, z_i + 1, ..., z_n) \quad \text{(functional substitutivity)}
\end{array}$$

A *computation* is a set of events, where $==$ is the equality operator on set elements.

The events in a computation have the preorder relation $\leq_c$ (causal preordering) and the preorder relations $\leq_t$ (temporal preordering, for each Clock t); and the equivalence relation, $=_c$ (causal equivalence) and the equivalence relations $=_t$ (temporal equivalence).

$=_c$ and $=_t$ are equivalence relations; that is, they satisfy the equivalence axioms of reflexivity, symmetry and transitivity (shown above). The relations $\leq_c$ and $=_c$ together satisfy the preorder axioms:

$$\begin{array}{lll}
(\forall \text{ event } e)e \leq_c e & & \text{(reflexivity)} \\
(\forall \text{ event } e_1, e_2)(e_1 \leq_c e_2 \wedge e_2 \leq_c e_1) & \rightarrow \quad e_1 =_c e_2 & \text{(antisymmetry w.r.t. } =_c\text{)} \\
(\forall \text{ event } e_1, e_2, e_3)(e_1 \leq_c e_2 \wedge e_2 \leq_c e_3) & \rightarrow \quad e_1 \leq_c e_3 & \text{(transitivity)} \\
(\forall \text{ event } e_1, e_2, e_3)(e_1 =_c e_2 & \rightarrow \quad e_1 \leq_c e_3) \quad \leftrightarrow \quad e_2 \leq e_3 & \text{(left substitutivity)} \\
(\forall \text{ event } e_1, e_2, e_3)(e_1 =_c e_2 & \rightarrow \quad e_3 \leq_c e_1) \quad \leftrightarrow \quad e_3 \leq e_2 & \text{(right substitutivity)}
\end{array}$$

$\leq_t$ and $=_t$ satisfy the same axioms. [1] **Sigurd thinks there's a cleaner and simpler set of axioms than these.**

From these relations we derive the associated relations $<_c$ (causal ordering) and $<_t$ (temporal ordering). $<_c$ is defined in terms of $\leq_c$ and $=_c$:

$$(\forall \text{ event } e_1, e_2) e_1 <_c e_2 \quad \leftrightarrow \quad (e_1 \leq_c e_2 \wedge \neg e_1 =_c e_2) \quad \text{(irreflexive restriction)}$$
$$(\forall \text{ event } e_1, e_2) e_1 \leq_c e_2 \quad \leftrightarrow \quad (e_1 <_c e_2 \vee \neg e_1 =_c e_2) \quad \text{(reflexive closure)}$$

$<_t$ is defined in terms of $\leq_t$ and $=_t$ with the same axioms.

The equivalence relations of time and causality are consistent with the identity relation (this relationship is inferrable from the congruence of $==$):

$$(\forall \text{ event } e_1, e_2) e_1 == e_2 \quad \rightarrow \quad e_1 =_c e_2$$
$$(\forall \text{event} e_1, e_2) e_1 == e_2 \quad \rightarrow \quad e_1 =_t e_2 \qquad \text{(identity-equality consistency)}$$

Causal ordering and temporal ordering have the following consistency relationship:

$$(\forall \text{ event } e_1, e_2) e_1 <_t e_2 \quad \rightarrow \quad \neg(e_2 <_c e_1)$$
$$(\forall \text{event} e_1, e_2) e_1 <_c e_2 \quad \rightarrow \quad \neg(t.finish(e_2) <_t t.start(e_1)) \quad \text{(causal-temporal consistency)}$$

1.3 The Pattern Language

A Computation $\mathcal{C}$ is a set of events.

We use the notation $\mathcal{C} \models P$ to mean pattern P matched in the computation $\mathcal{C}$. The result of matching a pattern in a computation is a set of sets of events. Every such set will be a subset of $\mathcal{C}$.

The pattern language is defined formally in Figure 1.1.

$\mathcal{C} \models a$	$\equiv \{\{e\} \mid (e \in \mathcal{C}) \wedge (a \in \mathcal{B}) \wedge matches(e, a)\}$	Basic Patterns[a], 2.3
$\mathcal{C} \models p_1 \longrightarrow p_2$	$\equiv \{s_1 \cup s_2 \mid (s_1 \in (\mathcal{C} \models p_1)) \wedge (s_2 \in (\mathcal{C} \models p_2)) \wedge$ $(\forall e_1, e_2\ (e_1 \in s_1 \wedge e_2 \in s_2) \rightarrow e_1 <_c e_2)\}$	Sequence, 2.4.1
$\mathcal{C} \models p_1 \rhd p_2$	$\equiv \{s_1 \cup s_2 \mid (s_1 \in (\mathcal{C} \models p_1)) \wedge (s_2 \in (\mathcal{C} \models p_2)) \wedge$ $s_1 \cup s_2 \in (\mathcal{C} \models p_1 \longrightarrow p_2) \wedge$ $(\forall e_1, e_2\ (e_1 \in s_1 \wedge e_2 \in s_2) \rightarrow$ $\neg \exists w\ (w \in \mathcal{C}) \wedge (w \notin s_1 \cup s_2) \wedge$ $e_1 <_c w <_c e_2\}$	Immediate Sequence, 2.4.2
$\mathcal{C} \models p_1 \sim p_2$	$\equiv \{s_1 \cup s_2 \mid (s_1 \in (\mathcal{C} \models p_1)) \wedge (s_2 \in (\mathcal{C} \models p_2)) \wedge$ $(s_1 \cap s_2) = \emptyset\}$	Join, 2.4.3
$\mathcal{C} \models p_1 \parallel p_2$	$\equiv \{\ s_1 \cup s_2 \mid (s_1 \in (\mathcal{C} \models p_1)) \wedge (s_2 \in (\mathcal{C} \models p_2)) \wedge$ $(\forall e_1, e_2\ (e_1 \in s_1 \wedge e_2 \in s_2) \rightarrow$ $\neg(e_1 <_c e_2 \vee e_2 <_c e_1))\}$	Independence, 2.4.4
$\mathcal{C} \models p_1$ **or** p_2	$\equiv (\mathcal{C} \models p_1) \cup (\mathcal{C} \models p_2)$	Disjunction, 2.4.5
$\mathcal{C} \models p_1$ **and** p_2	$\equiv (\mathcal{C} \models p_1) \cap (\mathcal{C} \models p_2)$	Conjunction, 2.4.6
$\mathcal{C} \models p_1 \cup p_2$	$\equiv \{s_1 \cup s_2 \mid (s_1 \in (\mathcal{C} \models p_1)) \wedge (s_2 \in (\mathcal{C} \models p_2))\}$	Union, 2.4.7
$\mathcal{C} \models p_1 \leftrightarrow p_2$	$\equiv \{s_1 \cup s_2 \mid (s_1 \in (\mathcal{C} \models p_1)) \wedge (s_2 \in (\mathcal{C} \models p_2)) \wedge$ $(\forall e_1, e_2\ (e_1 \in s_1 \wedge e_2 \in s_2) \rightarrow e_1 =_c e_2)$	Equivalence, **??**
$\mathcal{C} \models op(a_1, \ldots, a_n)$	$\equiv \mathcal{C} \models b_{op}(p_1, \ldots, f_n)\ \vert^{p_1,\ldots,f_n}_{a_1,\ldots,a_n}$	Pattern Macros [b], 2.5
$\mathcal{C} \models (id{:}t$ in it rel $op)f$	$\equiv$ *if* $l_{it} = [v_1, v_2, \ldots, v_n]$*then* $\mathcal{C} \models p \vert^{id}_{v_1}\ op\ p \vert^{id}_{v_2}\ op \ldots op\ p \vert^{id}_{v_n}$	Iteration [c], 2.4.8
$\mathcal{C} \models (id{:}t)p$	$\equiv (s \mid \exists v \in d_t (s \in \mathcal{C} \models p \vert^{id}_{v}))$	Placeholder Patterns [d], 2.4.9
$\mathcal{C} \models (p)$	$\equiv \mathcal{C} \models p$	Parenthesized Patterns, 2.4.10
$\mathcal{C} \models p$ where b	$\equiv$ *if* b *then* $(\mathcal{C} \models p)$ *else* $\{\}$	Guarded Patterns, 2.4.11
$\mathcal{C} \models p$ during (c, t_1, t_2)	$\equiv \mathcal{C} \models p \wedge (\exists e_1, e_2 \in \mathcal{C} \models p)(\forall e_3 \in \mathcal{C} \models p)$ $c.start(e_1) = t_1 \wedge$ $(c.start(e_1) <= c.start(e_3)) \wedge$ $c.finish(e_2) = t_2 \wedge$ $(c.finish(e_2) >= c.finish(e_3))$	Timing Operators [e], 2.6

[a] We assume the existence of a predicate $matches(a, e)$ which is true if a basic pattern $a \in \mathcal{B}$ matches an event $e \in \mathcal{C}$.

[b] $b_{op}(p_1, \ldots, f_n)$ is the pattern associated with the macro op, parameterized by its formal parameters.

[c] l_{it} is the set of objects denoted by it. Currently a bounded definition; need an unbounded one.

[d] d_t is the domain of type t; that is, the set of all values of t.

[e] $c.start(e)$ is the value of the Start function of clock c on the event e. $c.finish(e)$ is the value of the Finish function.

Chapter 2

Event Patterns

Event patterns are expressions that describe partial orders of events. A pattern acts as a "template" that may fit a partial order of events, if the events and the orderings between them satisfy the semantics of the pattern. The pattern language is used to describe *computations*. A computation is a set of events and the orderings between them.

A computation that fits a pattern is said to *match* the pattern.

A pattern is usually an incomplete description of a computation; there are several ways to leave "holes" in the description, which may be filled by any values. For example, *placeholders* are description holes such that all occurrences of a particular placeholder must be filled by the same value in a match.

Patterns are used three ways in RAPIDE: in *triggering* constructs, in *constraints*, and as *poset generator* patterns.

When occurring in a triggering construct—that is, a pattern matching statement of a process (see Executable LRM, Section 7.3), a behavior transition rule (see Architectures LRM, Section 2.6), a map rule (see Architectures LRM, Section 5.2.3), or as the left-hand side of a connection (see Architectures LRM, Section 3.4)—a pattern describes a subcomputation that is to be observed by the construct; once the construct observes that subcomputation, execution of the construct continues.

A pattern may also occur in a pattern constraint; see the Constraint LRM for the use of patterns in constraints.

Finally, a pattern may be used as a poset generator in a behavior (see Architectures LRM, Section 2.6), or as the right-hand side of a connection (see Architectures LRM, Section 3.4).

In this chapter, where the orderings of events in a computation are relevant and important, we will use figures to show the computation, such as in Figure 2.1. When such orderings are irrelevant, we will use simple set notation to give the set of events comprising the computation of interest. For example, { A, B, C } refers to a computation containing events A, B, and C, where the ordering of these events is not important.

2.1 Syntax

The syntax for patterns is

Figure 2.1: A computation

pattern ::= *basic_pattern* |
pattern binary_pattern_operator pattern |
iterator_decl pattern |
pattern **where** *guard* |
pholder_decl_list pattern |
'(' pattern ')' |
pattern_macro_instance

binary_pattern_operator ::= *'⟶'* | *'▷'* | *'∼'* | *'||'* | **or** | **and** | *'∪'*

iterator_decl ::= *'['* [*ident* **in**] *iterator_exp* *']'*

guard ::= *algebraic_constraint*

pholder_decl_list ::= *'('* *pholder_decl* { *';'* *pholder_decl* } *')'*

pholder_decl ::= *identifier_list* *':'* *type_expression*

The characters '−>' may be used as a replacement for '⟶'; '|>' may be used as a replacement for '▷'; and '**union**' may be used as a replacement for '∪'.

The character '^' followed by the binary_pattern_operator may be used as a replacement for the superscripted binary_pattern_operator in pattern iteration.

Guards have higher precedence than the other pattern operators, all of which have the same precedence. All non-pattern operators have higher precedence than guards.

In patterns, as in all expressions, operators of higher precedence are associated with their operands before operators of lower precedence. For a sequence of operators with the same precedence level, the operators are left associative. Parentheses can be used to impose other associations.

2.2 Placeholders

Patterns may contain occurrences of *placeholders*. These are identifiers that act similar to free variables. Placeholders represent "holes" in the patterns, where any value that is of the placeholder type may fit. Placeholders are fully defined in Section 2.4.9.

2.2.1 Syntax

pattern ::= *pholder_decl_list pattern*

pholder_decl_list ::= '(' *pholder_decl* { ';' *pholder_decl* } ')'
pholder_decl ::= *identifier_list* : *type_expression*

A placeholder declaration is followed by a pattern, which defines its scope.

2.2.2 Semantics

An occurrence of the placeholder in a pattern is an object expression of the placeholder's declared type.

For a particular match of a pattern, a placeholder scope defines a textual area of the pattern for which all occurrences of a placeholder must be matched by the same value, which must be of the placeholder's type.

2.2.3 Examples

Example: *A basic pattern*

$$(?D : \text{Dollar})\ \ \text{Deposit}(?D) \longrightarrow \text{Withdraw}(?D)$$

Commentary:

This pattern defines the placeholder ?D to be of the type Dollar[1]. A match of this pattern must be a Deposit event followed (as required by the $\longrightarrow$ pattern operator) by a Withdraw event, where both events must have the same parameter value.

□

2.3 Basic Patterns

The simplest patterns, *basic patterns*, are templates for a single event. Any match for such a pattern will be a computation consisting of one event.

2.3.1 Syntax

The syntax for a basic pattern is

basic_pattern ::= [*event_part*] '@' ['(' *parameter_association* ')']

basic_pattern ::= [*event_part* '@'] *action_part* ['(' *parameter_association* ')']

event_part ::= *expression*

action_part ::= [*expression* '.'] *ident*

[1] Stylistically, placeholders are represented in these LRMs as identifiers beginning with '?'. This style is recommended if the placeholders need to be distinguished from other identifiers in a pattern.

2.3.2 Semantics

The *event_part*, if it exists, is an expression of type **event**().

The *action_part* consists of an identifier, possibly preceded by an object expression, using the selection syntax.

Matching rules. A basic pattern

exp@(<parameters>)

is matched by a computation consisting of a single event, { E }, if the following hold:

- The parameters of the pattern are *consistent* with the parameters of the event type: meaning, for every positional parameter in the pattern, the event has a parameter in that position consistent with the pattern parameter; and, for every named parameter in the pattern, the event has a parameter of the same name that is consistent with the pattern parameter.

  A pattern's parameter is consistent with an event's parameter if they are of the same kind (that is, if the event's parameter is an object parameter and the pattern's parameter is an object expression, and similarly with type parameters), and where the type associated with the event parameter is of the type associated with the pattern parameter expression.

  Note: The positions of the predefined parameters of all events (see the RAPIDE 1.0 Predefined Types and Objects Reference Manual) are considered not available. That is, named association must be used to denote the value of a predefined event parameter.

- Each object parameter of the pattern is equal (i.e. has the "=" relationship) to its associated constituent in the event. If the type of the parameter does not define "=" the identity relationship "==" (see Chapter 22 of the RAPIDE 1.0 Predefined Types and Objects Reference Manual) is used.

  Therefore, if a parameter association for a particular parameter is omitted in a basic pattern, it may have any value.

- E is an event object that is identical (has the identity relationship) to the object denoted by exp, if an event part is given.

The basic pattern "@" matches any single event in a computation.

For a discussion of when object expressions in patterns are evaluated, see Section 3.4.

Alternative Syntax. An alternative syntax is also available for basic patterns; it has further static restrictions.

The basic pattern

exp@obj#id(<parameters>)

is matched by the computations that match

exp@(<parameters>, Name **is** "id", Owner **is** obj)

The basic pattern

exp@id(<parameters>)

is matched by the computations that match

exp@(<parameters>, Name **is** "id")

That is, the identifier supplied in the action part is interpreted as the string literal value of the Name parameter; the object expression of the action part, if any, is interpreted as the value of the Owner parameter.

The further restrictions are:

- If the action part expresses both the name and the owner, the name must denote an action in the owner's type; if the action part expresses only the name, that name must denote an action that is directly visible.
- The rules for parameter association and overload resolution are the same as those for function applications, except that an association for any particular parameter may always be omitted.
- The event type inferred by the parameter association (or by the empty parameter association, if none is given) must be a subtype of the type of the event part, if one exists.

2.3.3 Examples

The following examples use a banking system, where each action name indicates the set of transactions it models — Balance events report the balance of an account, Issue_Check represents the issuing of a check drawn on a particular account, Withdraw represents the withdrawal of money from a particular account, and Deposit represents cash deposits. The sums, accounts and check numbers involved are identified by the parameters of the events.

The type of Balance, Withdraw and Deposit events is
event(D: Dollar; A: Accnt)
The type of Issue_Check events is
event(D: Dollar; A: Accnt; N: Check_No)
Assume that Dollar, Accnt, and Check_No are Integer types.

Example:

@(235, Savings, Name **is** "Deposit")
Deposit(235,Savings)

Commentary:

The basic patterns above are equivalent; the second simply uses the more intuitive syntax.

Consider the computation of Figure 2.2, four events ordered in a sequence.

The second of the two Deposit events of the figure would constitute a match for the basic patterns, since only it has the action name given in the pattern, and its parameters are equal to those in the pattern.

Recall that the case of the Name string parameter is irrelevant: an equivalent basic pattern is @(235, Savings, Name **is** "DEPOSIT").

□

Example:

Deposit

Commentary:

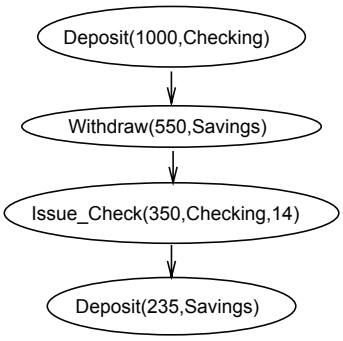


Figure 2.2: A simple computation (Arrows indicate causal dependency)

Each of the Deposit events of Figure 2.2 constitutes a match for this basic pattern. Because no parameters are given in the pattern, any parameter values are allowed.

The Issue_Check and Withdraw events would not match because their action names are different from the action name of the pattern.

□

Example: *A pattern using placeholders*

(?D : Dollar) Deposit(D **is** ?D, A **is** Savings)

Commentary:

This pattern declares and uses the placeholder ?D of type Dollar. It will be matched by a Deposit event with an A parameter equal to Savings, and any value for its D parameter. That value is the substitution value for ?D.

In the computation of Figure 2.2 only the second Deposit event would match the pattern.

□

Example: *An event part*

(?E : **event**(D: Dollar; A: Accnt)) ?E@Deposit

Commentary:

Each of the Deposit events of the figure would constitute a match for the pattern.

The substitution for ?E is a value that uniquely identifies the event matched. We can use the selection notation of record constituents to represent the parameter values of an event referred to by a placeholder; for example ?E.D would give us the D parameter value of the event represented by ?E. If the basic pattern were part of a larger pattern that contained other occurrences of ?E, all such occurrences would refer to the same event.

Section 2.4 on composite patterns show uses of identifying events with placeholders.

Example: *Using subtyping in matching*

```
type Privileged_Accnt is interface
    include Accnt;
    Priv_Code : Integer;
end interface;

(?PA : Privileged_Accnt) Deposit(A is ?PA)
```

Commentary:

Since ?PA is of type Privileged_Accnt, which is a subtype of Accnt, only a Deposit event that has a value of this subtype as its A parameter will match this pattern.

By subtyping of placeholders as well as by explicitly stating required parameter values, one can in most cases make sure a basic pattern is only matched by an event of interest, ignoring all others (another way to do this is by using *guards* — see Section 2.4.11). Subtype indications and parameter value requirements essentially function as simple *filters* in observing the computation.

□

Example: *A placeholder action part*

```
(?E : event(D: Dollar; A: Accnt)) ?E@
(?E : event(D: Dollar; A: Accnt) ?E@(D is 550)
```

Commentary:

The first pattern would match an event of that type (or a subtype of it) regardless of which action name the event is associated with. Thus each event of Figure 2.2 would be a match. Note that the type of the Issue_Check event is a *subtype* of **event**(D: Dollar; N: Check_No), since it has the required parameters (and others).

The second pattern would be matched only by those events that are of the type **event**(D: Dollar; A: Accnt) with a D parameter value of 550; ?E denotes the event.

In the computation of Figure 2.2 only the Withdraw event would match the pattern.

□

Example: *Performer of an action*

```
Deposit(Performer is Cust1)
(?C : Customer_Type) Deposit(Performer is ?C)
```

Commentary:

The first pattern will be matched by every Deposit event that has been performed by the Cust1 object; the second will be matched by every Deposit event that has been performed by an object of type Customer_Type.

2.4 Composite Patterns

One is often interested in recognizing complex patterns of behavior. Composite patterns are pattern expressions built from smaller patterns; while basic patterns are always matched by exactly one event, composite patterns may potentially be matched by any number of events.

The binary operation '$\longrightarrow$' on patterns is called the *sequence* combinator, '$\triangleright$' is called the *immediate sequence* combinator, '$\sim$' is called the *join* combinator, '||' is called the *independence* combinator, '**or**' is called the *disjunction* combinator, '**and**' is called the *conjunction* combinator, and '$\cup$' is called the *union* combinator.

An informal semantics for each combinator is given in the following sections.

2.4.1 Sequence

The sequence operator allows matching of two sets of events that are ordered in the dependency computation.

Semantics

A subcomputation C matches the sequence pattern

$$P_1 \longrightarrow P_2$$

if C can be partitioned into two subcomputations C_1 and C_2 such that (1) all events of C_1 precede all the events of C_2 in the dependency computation, and (2) C_1 matches P_1, and C_2 matches P_2.

Example: *A sequence pattern*

Deposit(A **is** Checking) $\longrightarrow$ Balance(A **is** Checking)

Commentary:

Assume Checking is declared as a value of type Accnt.

In the banking example, there may be many possible dependencies between deposits, withdrawals, and balance reporting. The balance of an account is dependent upon all the deposits and withdrawals that result in the particular balance value. Similarly, a withdrawal may be dependent upon a balance statement, in the sense that a person's decision to withdraw some amount may be influenced by whatever the balance statement says about the state of his account.

A subcomputation will match the above pattern if it consists of two events (one for each of the basic descriptors of the pattern), one matching the first basic pattern, the other the second, *and* they are ordered in the computation, with the Deposit event preceding the Balance event in the computation. Additionally, both must have the A parameter value of Checking.

In Figure 2.3 the two shaded events constitute a match for the pattern. However, the shaded Balance and the Deposit of $950 do not, since they are unordered. Neither does the Deposit for $100 together with the Balance constitute a match, since their ordering is the opposite of the one the pattern requires.

–

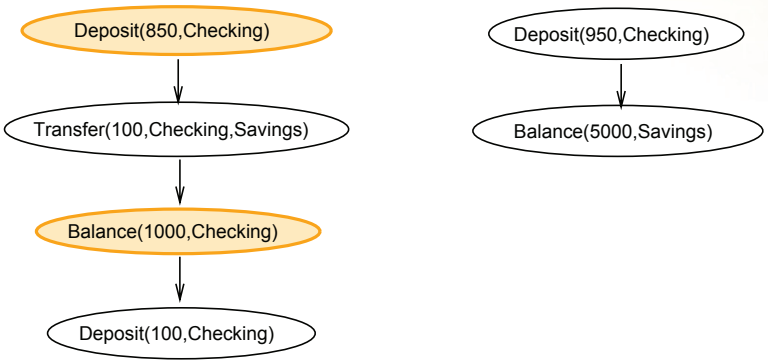


Figure 2.3: A match for a sequence pattern

2.4.2 Immediate Sequence

In addition to the sequence ordering of event patterns, it is often useful to require the immediate sequence ordering: two sets of events, one of which follows the other with no intervening events.

Semantics

A subcomputation C matches the sequence pattern

$$P_1 \triangleright P_2$$

if it can be partitioned into two subcomputations C_1 and C_2 such that (1) all events of C_1 precede all the events of C_2 in the dependency computation, (2) in the set of events S in which a match is sought (i.e. $C \subseteq S$), there exists no event that both follows any maximum of C_1 and precedes any minimum of C_2 in the dependency computation, and (3) C_1 matches P_1, and C_2 matches P_2.

This combinator refers to the entire set of events in which the match is sought; it is the only pattern combinator that does so. For patterns occurring in reactive statements of processes, see Section 7.3 of the Executable LRM; for patterns occurring in pattern constraints, see Section **??** of the RAPIDE 1.0 Constraint Language Reference Manual; and for poset generators, see Section **??** of the Executable LRM.

Example: *An immediate sequence pattern*

Deposit $\triangleright$ Balance

Commentary:

This pattern will be matched by two events, a Deposit followed in the ordering by a Balance, where no other visible events are between these two events in the ordering. In Figure 2.3, only the rightmost Deposit/Balance pair matches the pattern.

2.4.3 Join

The join operator combines two disjoint subcomputations that match the sub-patterns. In these LRMs we equivalently say that two disjoint computations are *joined* to form a composite computation, and that a computation is *partitioned* into two smaller computations.

Semantics

A subcomputation C matches the join pattern

$$P_1 \sim P_2$$

if C can be partitioned into two subcomputations C_1 and C_2 such that (1) C_1 matches P_1, and (2) C_2 matches P_2. In other words, disjoint C_1 and C_2 may be joined to form a match for the pattern.

Example:

Receive(?D1, His_Employer) $\sim$ Receive(?D2, Her_Employer)

Commentary:

All matches for this pattern will consist of two Receive events, one with the employer parameter being equal to the value of His_Employer, the other with the employer parameter being equal to the value of Her_Employer. Whether the two events are ordered does not matter.

If His_Employer=Her_Employer then a subcomputation consisting of only *one* Receive does *not* constitute a match for the pattern, since there is no partitioning of the singleton set which would result in two distinct matches for the two operands.

□

2.4.4 Independence

One often wants to determine not only the ordering of events, but also *whether* there is an ordering. Independence is a combinator requiring its constituent matches be unordered in the computation.

Semantics

A subcomputation C matches an independence pattern

$$P_1 \parallel P_2$$

if C can be partitioned into two subcomputations C_1 and C_2 such that (1) no event in C_1 is dependent upon any event in C_2 and vice versa, and (2) C_1 matches P_1 and C_2 matches P_2.

Note that since the definition requires a disjoint partitioning of the subcomputation, independence is not reflexive; that is, an event is not independent with itself.

Example: *Two independent events*

(?A : Accnt)
Deposit(A **is** ?A) || Deposit(A **is** ?A)

Commentary:

This pattern would be matched by two Deposit events that are unordered and that have the same A parameter value. The two shaded events of Figure 2.4 have this relationship.

□

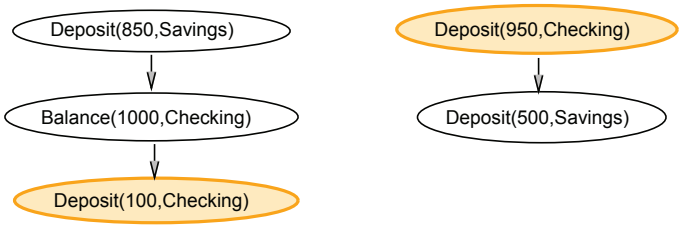


Figure 2.4: A match of two independent events

2.4.5 Disjunction

Instead of looking for a set of (sub-)patterns which all have to be matched for a match of the whole pattern to occur, we may equally well wish to look for a match of any *one* among a set of patterns.

Semantics

A subcomputation C matches a disjunction pattern

$$P_1 \ \textbf{or} \ P_2$$

if C matches P_1 or C matches P_2.

Example:

Withdraw **or** Balance

Commentary:

This pattern would be matched by a single event if that event is *either* a Withdraw event *or* a Balance event.

□

Example: *A disjunction of composite patterns*

(Balance(A **is** Checking) || Deposit(A **is** Checking))
or (Balance(A **is** Checking) $\longrightarrow$ Deposit(A **is** Checking))

Commentary:

This pattern would match pairs of Balance and Deposit events with A parameters equal to Checking where either: 1) the Balance event precedes the Deposit event in the computation, or 2) the two events are independent.

Figure 2.5 shows a computation with two matches for this pattern. In the first match, the Balance event that precedes the Deposit and the A constituent of which is equal to Checking, together with the Deposit form one match. The Deposit event and the Balance event that is independent with it form the other match.

If this pattern occurred in, for example, a process, either match could trigger continuation of the process. See Chapter 7 of the Executable LRM for a description of how a process chooses among more than one possible match.

□

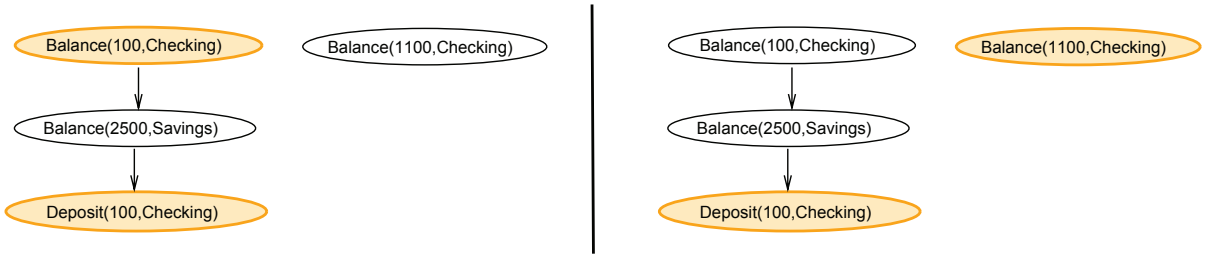


Figure 2.5: Matches of a disjunction pattern

2.4.6 Conjunction

We might want to describe a subcomputation that is a match for two different patterns. To do this, we use conjunction.

Semantics

A subcomputation C matches the conjunction pattern

$$P_1 \textbf{ and } P_2$$

if C matches P_1 *and* C matches P_2.

Example:

$$(\ (\text{Deposit} \longrightarrow \text{Withdraw}) \sim \text{Issue\_Check}\ ) \textbf{ and}$$
$$(\ (\text{Withdraw} \parallel \text{Issue\_Check}) \sim \text{Deposit}\ )$$

Commentary:

Here we state two requirements for a match of this pattern: a Deposit event must precede a Withdraw event, and a Withdraw event must be independent with an Issue_Check event. The pattern does not require or forbid an ordering between the Deposit and the Issue_Check. Figure 2.6 shows a computation that matches the pattern; again, if there were an ordering between the Deposit and the Issue_Check, the three events would still form a match.

–

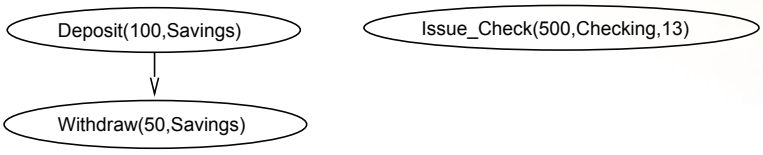


Figure 2.6: A match for event placeholders and conjunction

2.4.7 Union

We are sometimes interested in the union of two possibly overlapping sets of events, but not in the dependencies (if any) among them.

Semantics

A subcomputation C matches the union pattern

$$P_1 \cup P_2$$

if there are two (possibly overlapping) subcomputations C_1 and C_2 such that C_1 matches P_1 and C_2 matches P_2, and $C = C_1 \cup C_2$.

Example:

Receive(?D1, His_Employer) $\cup$ Receive(?D2, Her_Employer)

Commentary:

Assume His_Employer and Her_Employer are values identifying two persons' respective employers.

This pattern will be matched by a subcomputation consisting of either one or two Receive events.

Two Receive events, one with the employer parameter being equal to the value of His_Employer, the other with the employer parameter being equal to the value of Her_Employer, will be a match for the pattern. Whether or not the two events are ordered is irrelevant.

Additionally, if His_Employer=Her_Employer then a subcomputation consisting of only *one* Receive would constitute a match for the pattern as well, since there is no requirement that the matches for the two subcomputations be disjoint; ?D1 and ?D2 would in that case have the same substitution value.

When His_Employer=Her_Employer any two-event match would contain two other, one-event matches. For example, the subcomputation { Receive(100,Boss), Receive(200,Boss) } where His_Employer=Her_Employer=Boss contains a total of three matches of the above pattern: two of size one, and one of size two.

□

Example: *Union with event placeholders*

(?E@Deposit $\longrightarrow$ Withdraw) $\cup$ (?E@Deposit || Issue_Check)

Commentary:

This pattern will be matched by a Deposit event that is both followed by a Withdraw event and independent with an Issue_Check event. The relationship between the Withdraw event and the Issue_Check event is irrelevant: the two may or may not have an ordering. Figure 2.6 shows a computation that matches this pattern. The computation would also form a match if the Issue_Check preceded the Withdraw.

□

2.4.8 Iteration

One would often like to apply the binary pattern operations not only to two operands, but to any number of them. For example, we might want to match subcomputations with twelve Deposit events, or with an arbitrary number of ordered events.

Syntax

The binary pattern combinators may be iterated over via an iterator expression (See the Predefined Types LRM, Chapter 12), in which case from a pattern P, a pattern combinator O and an iterator that denotes n objects, we get a new pattern equivalent to n occurrences of P, related by the combinator O.

Instead of an iterator, one can use "*" and "+", which play roles similar to their namesakes in regular expression languages.

The syntax for iteration patterns is

pattern	::=	*iterator_decl pattern*
iterator_decl	::=	'[' [*ident* :] *iterator_exp* **rel** *binary_pattern_operator* ']'
iterator_exp	::=	*iterator* | '*' | '+'
iterator	::=	*object_expression* | *range*

Semantics

The *binary_pattern_operator* is either one of the predefined pattern combinators, or a pattern macro definition (see Section 2.5) with exactly two parameters, both of which are pattern parameters. The *iterator* is either an object expression that, for some type T, is a subtype of Iterator(T); or a range (see Predefined Types LRM, Section 12).

For a discussion of when object expressions in patterns are evaluated, see Section 3.4.

A subcomputation C matches the iteration pattern
[id : iter **rel** op] (P)
for a pattern operator *op*, an iterator expression *iter* that is a subtype of Iterator(T) for some type T, and a pattern P, in the following case.

The iterator expression implicitly defines a list of objects of type T of length n: $[L_1, L_2, \ldots, L_n]$

A list of patterns $P_1, P_2, \ldots, P_n$ is also implicitly defined where:

- $P_i \equiv P\,|_{L_i}^{id}$; that is, in P_i occurrences of id in the original pattern are replaced with an expression denoting L_i.

C matches the iteration pattern if and only if C matches

$$P_1 \; op \; P_2 \; op \; \ldots \; op \; P_n$$

If $n = 0$ (the iterator defines a list of length zero), the pattern is matched only by the empty computation. If $n = 1$, the pattern is matched by exactly those computations that match P.

A subcomputation C matches the pattern
[id : $*$ **rel** op] (P)
if C matches
(?n : Integer) [id : 1..?n **rel** op] ((P **where** ?n >= 0))

A subcomputation C matches the pattern
[id : $+$ **rel** op] (P)
if C matches
(?n : Integer) [id : 1..?n **rel** op] ((P **where** ?n > 0))

Iteration such as "[$*$ **rel** $\longrightarrow$] (P)" and "[$+$ **rel** $\longrightarrow$] (P)" is reminiscent of the "$*$" and "$+$" operators on regular expressions.

The pattern
[iter **rel** op] (P)
(i.e. an iterator pattern with no identifier) is matched by the computations that match
[unique : iter **rel** op] (P)
where unique is an unique identifier that does not occur in P. In this case, $P_1 \equiv P_2 \equiv \ldots \equiv P_n \equiv P$.

Example:

[1..12 **rel** $\sim$] (Deposit(A **is** Checking))

Commentary:

This pattern would be matched by 12 Deposits to the Checking account.

□

Example:

[$*$ **rel** $\longrightarrow$] (Balance)

Commentary:

This pattern describes zero or more ordered Balance events. A computation consisting of three ordered Balance events would actually contain eight matches for this pattern: one of size zero (the empty computation), three of size one, three of size two, and one of size three.

□

Example: *Placeholders in iterator patterns*

([1..?I **rel** ||] (Deposit)) $\longrightarrow$ Num_Deposits(?I)

Commentary:

Note that the iterator expression may contain placeholders.

This pattern would be matched by a Num_Deposits event, preceded by a number of independent Deposit events equal to the parameter of the Num_Deposits. There may be many distinct matches for this pattern in a given computation. For example, in a computation including the event Num_Deposits(3) which is preceded by five independent Deposit events, each of the $\binom{5}{3} = 10$ subcomputations of three Deposit events, along with the Num_Deposits event, will be a match for this pattern.

□

Example: *Named iterator*

[I : 1..10 **rel** —→] (Issue_Check(N **is** I))

Commentary:

The range 1..10 is an integer iterator, so the identifier I is of type Integer. Thus the pattern is matched by successive ordered matches of the basic pattern that follows; for each such match, the values of I are the successive values in the range.

That is, the above pattern is equivalent to the pattern

Issue_Check(N **is** 1) —→ Issue_Check(N **is** 2) —→ ...
—→ Issue_Check(N **is** 10)

□

Example: *Named iterator over a complex type*

[A : Airplane_Iter **rel** ||] ((A.Transmit_Position —→ Tower.Ack_Position(A)))

Commentary:

Here the expression Airplane_Iter is assumed to be an iterator over type Airplane. So identifier A is of the type Airplane, a type defining complex objects that that generate their own events. The pattern will be matched by a computation that consists of parallel matches of the pattern that follows the placeholder definition, where A has the value of an Airplane object for each match, and there is exactly one match for each value in the domain of Airplane.

Because the || operator is symmetric, the order defined by the iterator is irrelevant.

□

2.4.9 Placeholder Patterns

Placeholders may be declared in a pattern by giving a list of placeholder declarations, followed by a pattern. The scope of the placeholders is the pattern that follows the declarations.

Syntax

The syntax of a placeholder pattern is

pattern ::= *pholder_decl_list pattern*

pholder_decl_list ::= '(' *pholder_decl* { ';' *pholder_decl* } ')'

pholder_decl ::= *identifier_list* : *type_expression*

Semantics

The placeholder declaration defines a list of placeholders and their type. An occurrence of a placeholder in the pattern following its declaration is an object expression of the placeholder's type.

A subcomputation C matches a pattern

$$(I : T)P(I)$$

if there exists an object O of type T such that P(O) is matched by C.

For a particular match of a pattern, a placeholder scope defines a textual area of the pattern for which all occurrences of a placeholder must be matched by the same value. This value is called the *substitution value* of the match.

Often there will exist only one such substitution; for example, for a placeholder being matched to the argument of an event, the value of the argument will be the only possible substitution in that scope. If two occurrences of a placeholder are being matched to two different event arguments with different values, no substitution is possible, and so the subcomputation does not form a match. It is also possible for there to be more than one substitution (see the last example in Section 2.4.11).

There is no requirement that two distinct placeholders have distinct substitution values in a match.

Example: *Pattern-wide placeholder*

(?A : Accnt) [1..12 **rel** ∼] ((Deposit(A **is** ?A) ⟶ Deposit(A **is** ?A)))

Commentary:

The pattern will be matched by a computation C if it consists of twelve disjoint ordered pairs of deposit events, and all twenty-four deposit events have identical A parameter values.

□

Example: *Local placeholder definition*

[1..12 **rel** ∼] (((?A : Accnt) Deposit(A **is** ?A) ⟶ Deposit(A **is** ?A)))

Commentary:

The pattern will be matched by a computation C if it consists of twelve disjoint ordered pairs of deposit events, and the A parameters are equal *within each pair*. There are no restrictions on the values of A in different pairs.

□

Example: *Nested placeholder definitions with hiding*

(?A : Accnt)
Balance(A **is** ?A) ⟶
((?A : Accnt) Deposit(A **is** ?A) ⟶ [1..12 **rel** ∼] (Deposit(A **is** ?A))) ⟶
Balance(A **is** ?A)

Commentary:

The pattern will be matched by a computation C if it consists of (1) twelve disjoint ordered pairs of deposit events, and all twenty-four deposit events have the same A parameter values, (2) two Balance events respectively preceding and succeeding all the Deposit events, and (3) the A parameters of the two Balance events are equal.

There is no requirement that the A parameters of of the Balance events be equal to the A parameters of the Deposit events.

The ?A of the inner pattern could be replaced with a differently named placeholder of the same type without the meaning of the pattern changing. Indeed, to avoid confusion, nested definitions using the same placeholder identifier are not recommended.

□

2.4.10 Parenthesized Patterns

Syntax

The syntax of a parenthesized pattern is

'(' *pattern* ')'

Semantics

A subcomputation C matches a pattern

(P)

if it matches P.

Parenthesized patterns are used to associate sub-patterns.

2.4.11 Guarded Patterns

Context may be determined by various components of a pattern. For example, multiple occurrences of a placeholder must all have the same substitution value (see Section 2.4.9). Context may also be determined by the use of *guarded patterns*.

Syntax

pattern ::= *pattern* **where** *guard*
guard ::= *algebraic_constraint*

A guarded pattern contains a sub-pattern and a boolean expression, called the *guard*.

Semantics

A subcomputation C matches a *guarded pattern*

P **where** B

if C matches P and B has the value True.

For a discussion of how object expressions that occur in guarded patterns are evaluated, see Section 3.2.

Example: *A simple audit attention pattern*

(?D : Dollar)
Deposit(D **is** ?D) **where** (?D > 10_000)

Commentary:

An auditor process could be interested in deposits of checks whose amount exceeds $10_000. This pattern would be matched only by such events.

It is often useful (though not strictly correct) to consider ?D as "getting" its value from the Deposit event, and that value then being used in evaluating the guard.

□

Example: *Guarded composite pattern*

(?N1, ?N2 : Check_No)
(Issue_Check(N **is** ?N1) ⟶ Issue_Check(N **is** ?N2)) **where** ?N1 > ?N2

Commentary:

This pattern would be matched by two Issue_Check events where the checks are issued in reverse order. This could be of interest to a process auditing the banking transactions.

□

Example: *Using "=" in a guard*

(?D, ?Available : Dollar)
Balance(?D, Checking) **where** (?Available = ?D−Minimal_Balance) ⟶ ...

Commentary:

Because ?Available appears as an argument to the "=" operation, the value of the other argument is the only possible substitution value for ?Available. In the elided part of the pattern, the guard insures that other occurrences of ?Available are matched only to this value.

□

Example: *Bouncing checks—guards and state*

(?D : Dollar; ?A : Accnt)
Withdraw(D **is** ?D, A **is** ?A) **where** $(Balances(?A)) < ?D

Commentary:

Suppose there is an array Balances, where the array component Balances(n) reflects the balance of account n. The auditor could be looking for attempts to withdraw amounts exceeding the account balance — checks that would bounce — with the above pattern.

□

Example: *Multiple placeholder substitutions*

(?D : Dollar)
(Deposit(D **is** ?I) **or** Withdraw) **where** ?I > 0

Commentary:

This pattern is matched either by a Deposit event with Dollar argument greater than zero, or by any Withdraw event. In the case where a Withdraw event is the match, the placeholder ?D has multiple substitution values: any integer greater than zero would be a valid substitution. Thus the guard serves no useful purpose with respect to Withdraw, and may be moved to guard only the Deposit basic pattern.

□

2.5 Pattern Macros

One can use *pattern macros* to abstract (and parameterize) a pattern, and to define new operations on patterns.

2.5.1 Syntax

The syntax of these abstractions is

pattern_macro ::= **pattern** *identifier* '(' [*macro_parameter_list*] ')' **is** *pattern* ';'

macro_parameter_list ::= *macro_parameter* { ';' *macro_parameter* }

macro_parameter ::= *value_parameter* | *type_parameter* |
pattern *identifier_list*

pattern_macro_instance ::= *expression*

2.5.2 Semantics

A *pattern macro instance* is an application of parameters to a pattern macro. A subcomputation C matches a pattern macro instance if it matches the pattern obtained by taking the body of the macro, and replacing occurrences of the formal parameters by the actuals given in the instance.

Parameter conformance in a macro instance for value and type parameters follows the rules defined in Chapter 6 of the RAPIDE 1.0 Executable Language Reference Manual. The actual parameter associated with a parameter of the form **pattern** P must be a pattern expression. For some pattern *patt* passed as a parameter in a pattern macro instance, "(*patt*)" (i.e. the pattern parenthesized) is considered to be the substituted expression of the above definition; this avoids associativity problems.

Pattern macro declarations follow the rules for defining infix and prefix syntax, given in Chapter 4 of the RAPIDE 1.0 Executable Language Reference Manual. A pattern macro name may be used as a binary infix pattern operator if it has exactly two parameters. A unary prefix pattern operator may be defined if it is one of the predefined unary operators, and the macro has exactly one parameter.

Pattern macros serve several purposes. They may act as reusable patterns. They may also simplify patterns by breaking them up into smaller units. Recursive macros allow the definition of patterns that are otherwise inexpressible. Finally, new pattern operators can be defined using macros.

Recursive macros and matchability. Note that recursive pattern macro definitions are possible. Some such macros will be inherently unmatchable, e.g.:

```
pattern P() is P(); -- Macro calls itself
```

Others may be matchable, depending on the strength of a particular toolset:

```
pattern P() is
    P() or A where Fermats_Last_Thm_Is_True();
```

A toolset is not required to support all patterns that may be matchable. Further, matchability may not necessarily be detected by a compiler or run-time system.

Pattern macro profile. More than one pattern macro with the same name may be defined in a declarative region as long as the macros do not have the same *profile*. The profile of a pattern macro consists of: the number of parameters the macro has; whether the parameter is a type-, object-, or pattern-parameter; and typing information if it is a type- or object-parameter.

Two pattern macros have the same profile if they have the same number of parameters, and if, positionally: where one macro has a type parameter, the other also has a type parameter (further, if either has a type constraint, the other has a type constraint of an equivalent type); where one has a pattern parameter, the other also has a pattern parameter; and where one has an object parameter, the other has an object parameter of an equivalent type.

2.5.3 Examples

Example: *Defining a pattern operator*

```
pattern Optional( pattern P ) is P or Empty();
```

Commentary:

The macro Optional defines a new unary pattern operator. The semantics of Optional are that, given some pattern P, Optional(P) is equivalent to the pattern (P) **or** Empty().

□

Example: *A recursive pattern macro definition*

```
pattern Ticks() is Tick ⟶ Ticks();
```

Commentary:

Assume Tick is an action name. The pattern "Ticks()" is an instance of the above macro. It describes an infinite chain of ordered Tick events. Pattern macros permit the description of infinite computations, which are otherwise inexpressible (iteration can only describe finite, though arbitrarily large, computations). An infinite pattern will, of course, have limited use in RAPIDE processes, but can be useful in pattern specifications.

□

Example: *A bounded recursive macro*

```
pattern Bounded_Ticks( I : Integer ) is
    (Tick(I) where (I <= 20) ⟶ Bounded_Ticks(I+1))
      or
    Empty() where (I > 20);
```

Commentary:

Assume Empty is a pattern macro, instances of which are matched by zero events.

An instance of this macro is a pattern matched by a subcomputation consisting of a sequence of Tick events with incrementally increasing parameter values, which terminates whenever a Tick with parameter 20 is encountered. For example, Figure 2.7 shows a computation that matches the pattern Bounded_Ticks(18). Instances of Bounded_Ticks with a parameter greater than 20 will be matched only by the empty computation.

□

2.6 Timing Operator

The Timing operator is a predefined macro that allow patterns to refer to the time at which events matched by the patterns are generated.

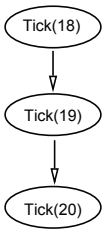


Figure 2.7: A match for a macro instance

Several other related pattern time operators are defined in terms of Timing; they are defined in the Predefined Types LRM, Chapter 18.

pattern Timing(**pattern** P; Time, Duration : Integer; L : Clock) **is**
implementation_defined

The pattern Timing(P,T,D,L), where T and D are of type Integer (a subtype of all Ticks types) and L is of type Clock, is matched by a computation C such that (1) C matches P, and (2) all events in C are related to L, and (3) T is the minimum (earliest) C.Start value of any event in C, and D is the duration of the pattern, calculated by taking the maximum (latest) C.Finish value of any event in C, minus T.

2.6.1 Examples

Example:

(?First, ?Dur : Integer)
Timing((Start_Proc1 || Start_Proc2) —→ End_Proc, ?First, ?Dur, C)

Commentary:

This pattern will be matched by computations consisting of three events, a Start_Proc1, Start_Proc2, and End_Proc event. Where S1 is the event matched to Start_Proc1 and S2 is the event matched to Start_Proc2, ?First will have the substitution value of C.Start(S1) or C.Start(S2), whichever is smaller. Where E is the event matched to End_Proc, ?Dur will have the substitution value of C.Finish(E) − ?First.

Chapter 3

Program State and Patterns

The concept of "state" in a concurrent program is more complex than it is for sequential programming languages. Because processes execute independently, the concept of a single sequence of program states does not apply. Similarly, several different processes may be independently accessing program data (in particular, dereferences of reference objects).

The result of a RAPIDE simulation is a partially ordered set of events; hence the state of a RAPIDE program is defined in terms of the events produced. State is defined with respect to a single event, or a set of events.

This definition of state enables us to define the meaning of reference object dereferences that occur in patterns.

3.1 State

A *consistent cut* of a computation C is a subset C_1 such that no event in $C - C_1$ precedes any event in C_1. (*i.e.*, one may think of the cut as a line across the computation, with all the causal arrows crossing the line doing so in the same direction. C_1 is then the set of events before the line. A different phrasing of the same definition: A consistent cut is a prefix-closed subset of C.)

A consistent cut *at* an event e is any consistent cut where e is a maximum of the cut. For a given event e, there may be more than one such consistent cut.

Observe that a consistent cut of the computation C identifies a set of modules—namely those modules that were generated before the cut (*i.e.*, those modules whose start event is in the cut). Furthermore, a consistent cut identifies the value associated with a reference object, namely the parameter of the last assignment to the reference object.

A *state* is then a set of objects and the dereference values (of the reference objects) determined by a consistent cut.

A *state at an event* e is a state determined by a consistent cut at e. Note that there may be more than one such state associated with an event.

A *state at a set of events* P is a state determined by a consistent cut at a maximum of P such that P is a subset of the cut. Note that there may be more than one such state associated with a set of events.

3.2 State in Guards

Guarded patterns are defined in Section 2.4.11.

Recall that a subcomputation C matches

P **where** B

if C matches P and B has the value True. B is evaluated in some state at the set of events that comprises the match.

The evaluation of B does not by itself have any effect on the computation. In the evaluation, attempts to perform assignments to reference objects not created by the evaluation itself raises the exception Illdefined_Guard.

3.2.1 Commentary

There are two interesting properties of guards using state.

Consistent Semantics in Pattern Matching and Constraint Checking: If a region contains a trigger

P **where** B

and a constraint

never P **where** B;

then if the trigger ever fires, a constraint violation must also occur.

Repeatable Reads: Dereferencing the same reference object multiple times in a guard will always return the same value. This is called repeatable reading; dereferencing (reading) a variable repeatedly in a single guard returns the same value.

Note that this does not hold in a statement, for example: two dereferences in a single statement may return different values, if other processes are independently assigning to that object.

3.3 Examples

Example: *State expression in a process*

```
when (?D in Dollar, ?A in Accnt)
        Withdraw(?D, ?A) where ?D > Balance(?A) do
    -- Take overdraft action
end when;
```

Commentary:

Assume Balance is a function returning the balance of various accounts. Here the value of Balance(?A) is not determined until the pattern is being matched. If a Withdraw event is generated during the execution of this process body, Balance(?A) for the new event is not evaluated until the body execution is complete, and the pattern is again matched.

Example: *State expression in a process*

```
never (?D in Dollar, ?A in Accnt)
        Withdraw(?D, ?A) where ?D > Balance(?A);
```

Commentary:

In this constraint, Balance(?A) is evaluated as soon as a Withdraw event is generated and matches the guarded pattern.

□

3.4 References to State Outside of Guards

A pattern containing dereferences of Ref objects (where those dereferences do not appear in a guard expression) is equivalent to a pattern in which all Ref object derefereces appear in a guard expression.

3.4.1 Examples

Example: *State reference in a basic pattern parameter*

```
E($expr)
(?P in T) E(?P) where ?P = $expr
```

Commentary:

Where "expr" is some Ref object expression of type Ref(T), the above patterns are equivalent.

□

Example: *State reference in an iterator expression*

```
[ 1. .$expr rel —→] (A)
(?P in T) [ 1. .?P rel —→] (A) where ?P = $expr
```

Commentary:

The above patterns are equivalent.