SAVARA 1.1

# User Guide

by Gary Brown and Jeff Yu

**1. Overview** ........ 1
1.1. WS-CDL ........ 1
1.2. pi4soa ........ 2
1.3. SOA Lifecycle Governance ........ 2
1.3.1. Design Time Governance ........ 2
1.3.2. Runtime Governance ........ 3
1.4. First Steps ........ 4
**2. BPEL** ........ 5
2.1. Overview ........ 5
2.2. Generating a BPEL process ........ 5
2.2.1. Overview ........ 5
2.2.2. Generating the BPEL Process from a Choreography Description ........ 5
2.2.3. Limitations with the current CDL to BPEL mapping ........ 6
**3. Conversation Validation with CDL** ........ 7
3.1. Overview ........ 7
3.2. Configuration of Conversation Validation ........ 7
3.2.1. Installing the Conversation Validation Mechanism ........ 7
3.2.2. Explicit Configuration ........ 9
3.2.3. Defining the Validator Configuration within a Choreography ........ 10
3.3. Monitoring the Choreography Description ........ 12
3.4. Configuration for Conversation Recording ........ 14

# Overview

The SAVARA project aims to leverage the concept of a choreography (or conversation) description to provide design-time and run-time governance of an SOA.

A Choreography provides the means to describe the service interactions between multiple parties from a global (or service neutral) perspective. This means that it is possible for an organisation to define how an end-to-end business process should function, regardless of whether orchestrated or peer-to-peer service collaboration will be used.

Although in simple situations, a BPEL process description can provide a description of the interactions between multiple services, this only works where a single orchestrating process is in control. The benefit of the choreography description is that it can be used to provide a global view of a process across multiple orchestrated service domains.

This document will outline how the Choreography Description is being used as part of SAVARA to provide SOA governance capabilities for each phase of the SOA lifecycle.

When a validated design has been approved by the users, it can be used to generate an initial skeleton of the implementation for each service. The current version of SAVARA enables a skeleton implementation to be generated as a service implementation (e.g. WS-BPEL process).

## 1.1. WS-CDL

WS-CDL, or Web Service Choreography Description Language, is a candidate recommendation from W3C. Although associated with W3C and Web Services, it is important to begin by stating that the Choreography Description Language (CDL) is **not** web service specific.

The purpose of CDL is to enable the interactions between a collection of peer to peer services to be described from a neutral (or global) perspective. This is different to other standards, such as WS-BPEL, that describe interactions from a service specific viewpoint.

In essence a choreography description declares roles which will pass messages between each other, called interactions. The interactions are ordered based on a number of structuring mechanism which enables loops, conditional, choices and parallelism to be described. In CDL variables used for messages and for conditionals are all situated at roles. There is no shared state rather there is a precise description of the state at each role and a precise description of how these roles interact in order to reach some notion of common state in which information is exchanged and processed between them.

In CDL we use interactions and these structuring mechanisms to describe the observable behaviour, the messages exchanges and the rules for those exchanges and any supporting observable state on which they depend, of a system.

## 1.2. pi4soa

*pi4soa* is an open source project established to demonstrate the potential benefits that a global model (as described using CDL) can provide when building an SOA. The open source project is managed by the Pi4 Technologies Foundation, which is a collaboration between industry and academia.

Building complex distributed systems, without introducing unintended consequences, is a real challenge. Although the Choreography Description Language provides a means of describing complex systems at a higher level, and therefore help to reduce such complexity, it does not necessarily guarantee that erronous situations cannot occur due to inappropriately specified interactions. The research, being carried out by members of the Pi4 Technologies Foundation, into the global model and endpoint projection is targeted at identifying potential unintended consequences, to ensure that a global description of a system can be reliably executed and can be free from unintended consequences.

The tool suite currently offers the ability to:

- Define a choreography description
- Export the description to a range of other formats, such as BPMN, UML activity/state/sequence models, and HTML
- Define scenarios (equivalent to sequence diagrams), with example messages, which can then be simulated against an associated choreography

## 1.3. SOA Lifecycle Governance

### 1.3.1. Design Time Governance

Design-time governance is concerned with ensuring that the resulting system correctly implements requirements (whether functional or non-functional). A choreography description can be used to ensure that the implemented system meets the behavioural requirements.

The behavioural requirements can be captured as a collection of scenarios (e.g. sequence diagrams) with associated example messages. This enables an unambiguous representation of the business requirements to be stored in a machine processable form, which can subsequently be used to validate other phases of the SOA lifecycle.

Once the choreography description for the SOA has been defined, it can be validated against the scenarios, to ensure that the choreography correctly handles all of the business requirements.

Once the service enters the implementation phase, it is important to ensure that it continues to adhere to the design and therefore meets the business requirements. Currently this is achieved through the use of techniques such as continuous testing. However this is only as reliable as the quality of the unit tests that have been written.

When a 'structured' implementation language has been used, such as WS-BPEL, it will be possible to infer the behaviour of the service being implemented, to compare it against the choreography description.

Detecting incorrectly implemented behaviour at the earliest possible time saves on downstream costs associated with finding and fixing errors. By using static validation against the original design, it ensures that the implemented service will deliver its expected behaviour first time. This is important in building large scale SOAs where different services may be implemented in different locations.

There are two other areas where a choreography description can be used as part of design-time governance, that are not currently implemented in SAVARA:

- Service lookup – the choreography description can be used to determine if a service already exists in the Service Repository that meets the appropriate behavioural requirements.

- Service unit testing - this can be achieved using the scenarios originally specified to document the behavioural requirements. Rather than develop an independent source of test data, the scenarios can be used to validate the sequence of messages sent to, and received from, a service, as well as validating the contents of the messages returned from the service under test.

## 1.3.2. Runtime Governance

Runtime governance ensures that the SOA executes as expected according to predefined policies. In this context, a choreography description can be used in two ways.

### 1.3.2.1. Service validator

The choreography description represents the interactions between multiple services to deliver a business goal. To validate the behaviour of each individual service, within the choreography description, the behaviour of each service can be derived from the choreography.

The derived behaviour (or “endpoint projection”) of a service can be used within a 'service validator' to monitor the inbound and outbound messages for the service, to ensure they conform to the expected behaviour. If an invalid message is detected, it would be possible to block it, to prevent it from causing subsequent problems in downstream systems. The error can also be reported to a central management capability.

The SAVARA Validator provides the ability to configure service validators to monitor the behaviour of individual services. An enhanced version of the JBossESB trailblazer example has been included, with the appropriate validator configuration, to demonstrate this mechanism.

### 1.3.2.2. Process correlation

Validating each service locally can enable errors to be detected quickly, and the effects of the error prevented from contaminating other systems by blocking the erroneous messages.

However local service specific validation may not be adequate to identify errors that would affect the end-to-end business process. Therefore the message activity at each service validator can be reported to a central 'process correlation engine' which can reconstitute a global view of the business transaction, and determine if it matches the expected behaviour as defined in the choreography description.

The benefit of a correlated global view of the distributed business transaction is that it can be further analysed to ensure other governance polices have been followed – e.g. SLAs.

The pi4soa tool suite includes a simple GUI based monitoring tool to display the information obtained from correlating message events associated with individual services. The trailblazer example has been written to cause out of sequence messages under certain circumstances. See the “Samples Guide” for more information on how to run this example.

## 1.4. First Steps

The first step will be to follow the instructions in the Getting Started Guide to install SAVARA.

Once installed, the next step should be to try out the examples in the samples folder. The examples consistent of:

- Choreography related examples
  These examples provide an illustration of how to use scenarios, choreographies and other associated artifacts.

- Service Validation related examples
  The samples folder contains an enhanced version of the trailblazer example from the JBossESB, with the addition of a File Based Bank, and message content including a conversation id to enable the messages to be correlated with a specific session.

# BPEL

## 2.1. Overview

This section will describe generation and conformance checking features related to WS-BPEL.

This initial release provides basic support for generating BPEL processes from a choreography description. Subsequent releases will also provide conformance checking, to ensure that changes to a BPEL process are validated to ensure the process remains conformant with the choreography.

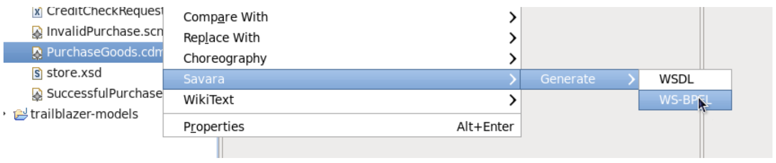
## 2.2. Generating a BPEL process

### 2.2.1. Overview

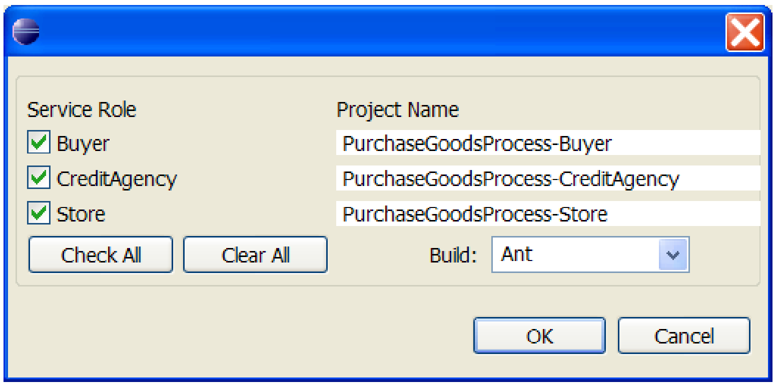
This section explains how to generate a template BPEL process from architectural and design artifacts.

### 2.2.2. Generating the BPEL Process from a Choreography Description

When the choreography description has been completed, and has no errors, the user should select the "SAVARA->Generate->WS-BPEL" menu item from the popup menu associated with the choreography description (.cdm) file.



When the dialog window is displayed, it will contain the list of services that can be generated, along with the project names that will be created. The user can unselect the services they do not wish to generate (also using the 'Check All' or 'Clear All' buttons).



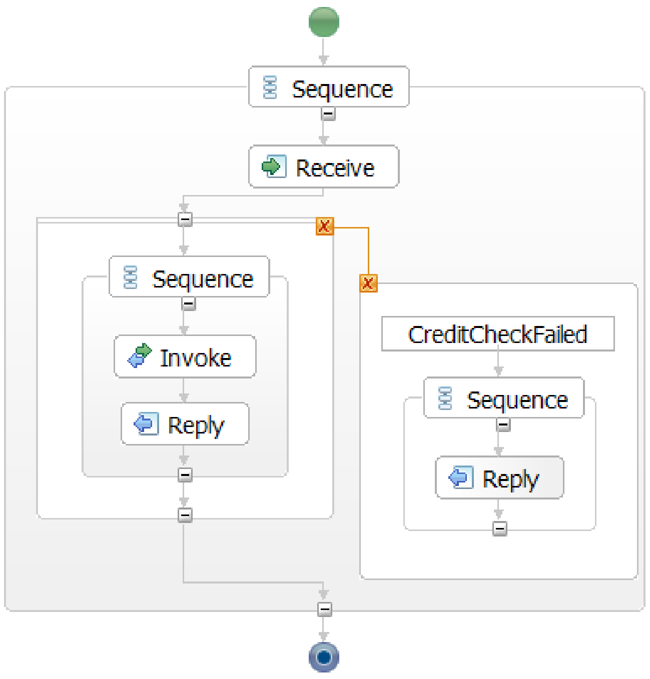
| Service Role | Project Name |
|---|---|
| ✓ Buyer | PurchaseGoodsProcess-Buyer |
| ✓ CreditAgency | PurchaseGoodsProcess-CreditAgency |
| ✓ Store | PurchaseGoodsProcess-Store |

Build: Ant

The user can also select their preferred build system, which will create the relevant build structure.

If there is a problem with the name of the project select, such as invalid characters used in the name, or the project name already exists, then it will be displayed in red.

Once the BPEL is generated, it can be viewed using the Eclipse BPEL editor, e.g.



## 2.2.3. Limitations with the current CDL to BPEL mapping

This initial version of the BPEL generation is primarily targeted at generating the interactions and grouping constructs. These are the important components that will be required when doing conformance checking as part of the next milestone.

This means that assignments and conditional expressions are not currently generated. The 'when' construct in CDL (also known as the blocking workunit) is also not currently handled. This may possible be implemented using flow links.

# Conversation Validation with CDL

## 3.1. Overview

Conversation validation is a form of runtime governance concerned with the dynamic behaviour of a system.

When coupled with a choreography description model of a system, this means having the ability to ensure that the way a collection of services interact correctly adheres to a description of the business process being enacted.

This section introduces the choreography description language (CDL) defined by W3C, and the *pi4soa* open source project which provides an editor for creating choreography descriptions, as well as utilizing these descriptions for runtime validation and execution purposes.

## 3.2. Configuration of Conversation Validation

This section explains how to configure the conversation validation mechanism to validate services against a choreography description. The first sub-section describes how the mechanism is hooked into the JBossESB and JBossWS-Native environments. The following two sub-sections explain two alternate ways that relevant endpoint references can be configured for validation.

### 3.2.1. Installing the Conversation Validation Mechanism

#### 3.2.1.1. JBossWS-Native

The principle mechanism used for validating conversations within the JBossWS Native stack is through the use of a global filter registered in the `standard-jaxws-client-config.xml` and `standard-jaxws-endpoint-config.xml` files. These files are located in the *$JBossAS/server/default/deployers/jbossws.deployter/META-INF* folder.

The `standard-jaxws-client-config.xml` is updated to include a
<emphaiss>pre handler</emphaiss>
implemented by a Savara client interceptor.

```
  <client-config>
    <config-name>Standard Client</config-name>
    <pre-handler-chains>
      <javaee:handler-chain>
        <javaee:protocol-bindings>##SOAP11_HTTP</javaee:protocol-bindings>
        <javaee:handler>
       <javaee:handler-name>SAVARA JBossWS-Native Client Validator Interceptor</javaee:handler-
name>
                                                                               <javaee:handler-
class>org.jboss.savara.validator.jbosswsnative.JBossWSNativeClientInterceptor</javaee:handler-
class>
        </javaee:handler>
      </javaee:handler-chain>
    </pre-handler-chains>
    <feature>http://org.jboss.ws/dispatch/validate</feature>
    <property>
      <property-name>http://org.jboss.ws/http#chunksize</property-name>
      <property-value>2048</property-value>
    </property>
  </client-config>
```

The `standard-jaxws-endpoint-config.xml` is updated to include a <emphaiss>pre handler</emphaiss>
implemented by a Savara server interceptor.

```
  <endpoint-config>
    <config-name>Standard Endpoint</config-name>
    <pre-handler-chains>
      <javaee:handler-chain>
        <javaee:protocol-bindings>##SOAP11_HTTP</javaee:protocol-bindings>
        <javaee:handler>
          <javaee:handler-name>Recording Handler</javaee:handler-name>
              <javaee:handler-class>org.jboss.wsf.framework.invocation.RecordingServerHandler</
javaee:handler-class>
        </javaee:handler>
      </javaee:handler-chain>
      <javaee:handler-chain>
        <javaee:protocol-bindings>##SOAP11_HTTP</javaee:protocol-bindings>
        <javaee:handler>
       <javaee:handler-name>SAVARA JBossWS-Native Service Validator Interceptor</javaee:handler-
name>
                                                                              <javaee:handler-
class>org.jboss.savara.validator.jbosswsnative.JBossWSNativeServerInterceptor</javaee:handler-
class>
        </javaee:handler>
      </javaee:handler-chain>
    </pre-handler-chains>
  </endpoint-config>
```

These interceptors are installed as part of the installation process for the SAVARA distribution.

### 3.2.1.2. JBossESB

The principle mechanism used for validating conversations within an ESB is through the use of a global filter registered with the *jbossesb-properties.xml*. This file is located in the *$JBossAS/server/default/deployers/esb.deployer* folder.

```
        <properties name="filters">
            ...
            <property name="org.jboss.soa.esb.filter.10"
                    value="org.jboss.savara.validator.jbossesb.ValidatorFilter"/>
        </properties>
```

This filter is installed as part of the installation process for the SAVARA distribution.

### 3.2.2. Explicit Configuration

The information concerning which destinations will be validated, and to which model/role they relate, can be explicitly defined within the *validator-config.xml* file, contained within the *savara-validator-jbossesb.esb* bundle.

An example of the contents of this file, that would related to the TrailBlazer example, is:

```
<validator mode="monitor" replyToTimeout="10000" >
    <service model="TrailBlazer.cdm"
                role="LoanBrokerParticipant" >
        <output epr="jms:queue/esb-tb-creditAgencyQueue" />
        <input epr="jms:queue/esb-tb-creditAgencyQueue_reply" />
        <output epr="jms:queue/esb-tb-jmsBankRequestQueue" />
        <output epr="jms:queue/esb-tb-fileBankRequestQueue" />
        <input epr="jms:queue/esb-tb-jmsBankResponseQueue" />
        <output epr="jms:queue/esb-tb-customerNotifier" />
        <input epr="jms:queue/esb-tb-fileBankResponseQueue" />
    </service>
    <service model="TrailBlazer.cdm"
                role="CreditAgencyParticipant" >
        <input epr="jms:queue/esb-tb-creditAgencyQueue" />
        <output epr="jms:queue/esb-tb-creditAgencyQueue_reply" />
    </service>
    <service model="TrailBlazer.cdm"
                role="BankParticipant" >
        <input epr="jms:queue/esb-tb-jmsBankRequestQueue" />
        <input epr="jms:queue/esb-tb-fileBankRequestQueue" />
        <output epr="jms:queue/esb-tb-jmsBankResponseQueue" />
        <output epr="jms:queue/esb-tb-fileBankResponseQueue" />
    </service>
    <service model="TrailBlazer.cdm"
                role="NotifierParticipant" >
        <input epr="jms:queue/esb-tb-customerNotifier" />
    </service>
</validator>
```

The 'validator' element has an optional attribute called 'mode', with the possible values of 'monitor' or 'manage'. If the mode is 'monitor' (which is the default), then any messages that result in validation errors being detected will continue to be received or sent, with the errors only be reported for information purposes. If the mode is 'manage', then any erronous messages detected during validation, that conflict with the behaviour as described in the choreography, will be prevented from being received or sent.

**Note**

It is important to note that if 'manage' validation mode is used, then the validation mechanism will be an integral part of the message flow. This may have a slight performance impact on the delivery of messages between services.

The optional 'replyToTimeout' (defined in milliseconds) is used to determine how long a dynamic reply-to destination should be monitored for validation purposes. In some message exchanges, the response destination will not always be known in advance. Therefore the configuration can identify such situations, and monitor the reply-to destination for the response. However, if a response is not delivered in a particular time period, we need to be able to discontinue the validation of the dynamic endpoint. If this did not occur, then over time too many endpoints would be monitored, which may result in out-of-memory problems. The default timeout period is 10 seconds.

Within the 'validator' element is a list of 'service' elements, one per service being validated. The behaviour of the service being validated is identified by specifying the model (e.g. choreography description file) and the role (e.g. participant type) within the model. Therefore, within the above configuration, the first set of destinations (eprs) are associated with the *LoanBrokerParticipant* defined within the choreography description model found in the file `TrailBlazer.cdm`, which will be located within the `models` folder contained within the *savara-validator-jboss.sar* bundle.

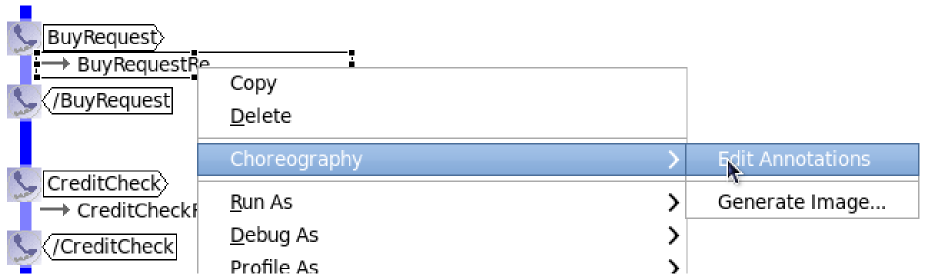
The elements contained within the 'service' element define the *input* and *output* eprs (Endpoint References) that are associated with the service. The *input* eprs are the destinations on which messages will be received and the *output* eprs are the destinations on which messages will be sent by the service.

The format of the 'epr' attribute will be specific to the type of transport being validated. Currently JMS is supported, and can be identified by the protocol prefix 'jms:', or a Web Service endpoint using a service name with the QName style of '{namespace}localpart'.

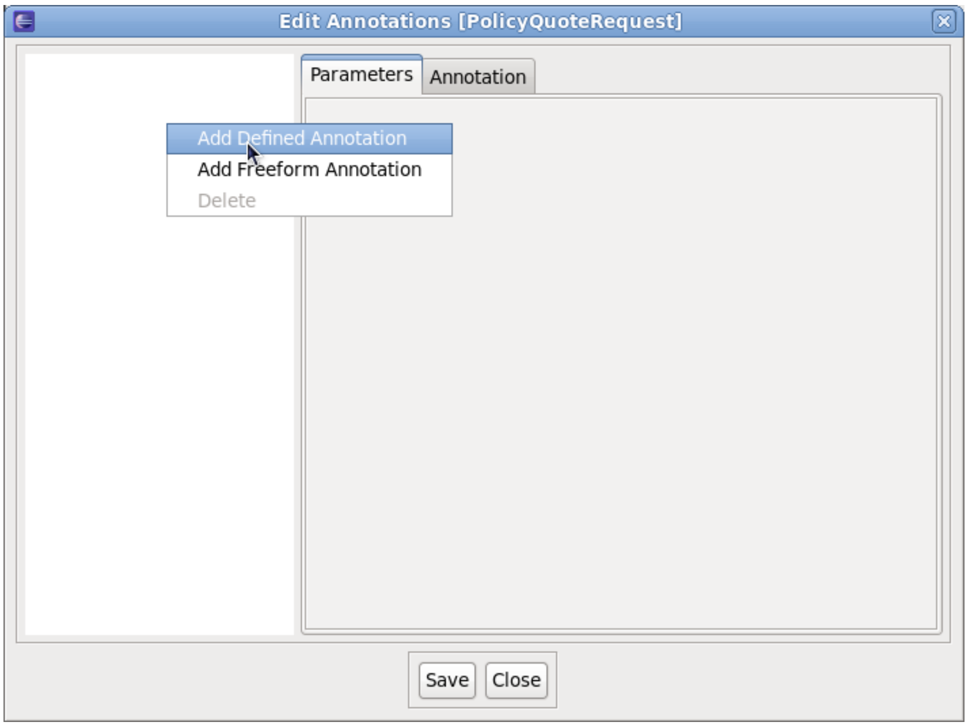
Each 'input' and 'output' element can also define an optional 'dynamicReplyTo' boolean attribute. If defined, it will indicate to the Service Validator that the message on the specified endpoint (epr) will contain a dynamically defined 'reply-to' destination that needs to be monitored for a response.

## 3.2.3. Defining the Validator Configuration within a Choreography

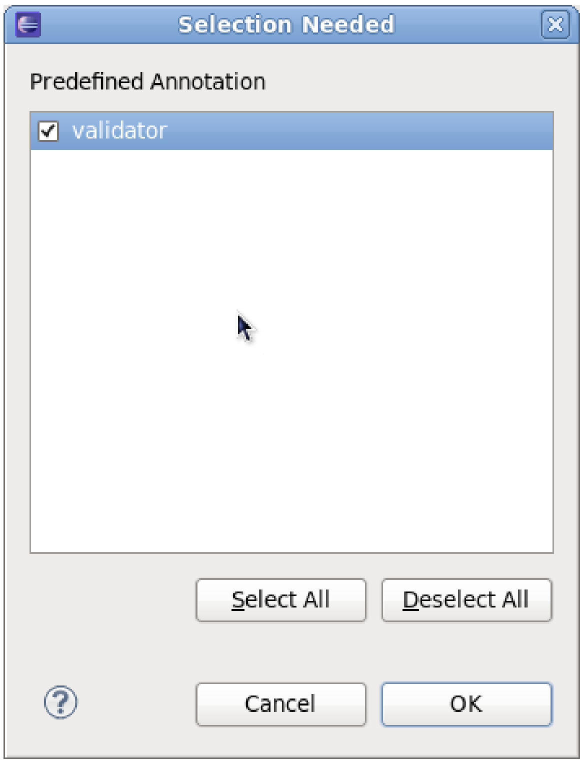
The first step to configuring the validator is to associate the endpoint references (EPRs) against the relevant choreography interactions. This is achieved by defining an annotation for each 'exchange details' component (i.e. each request and response/notification).



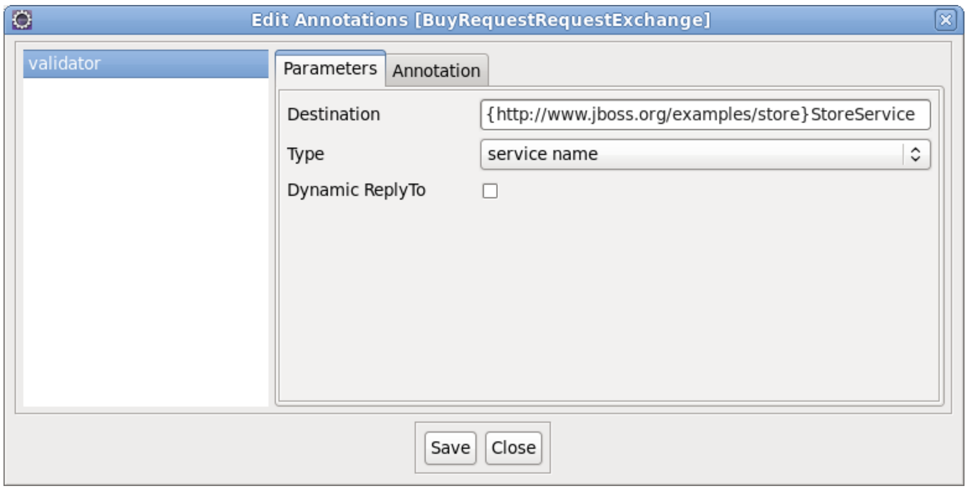
When the annotation editor is displayed for the relevant 'exchange details' component, the *validator* annotation should be added. This is achieved by selecting the popup menu associated with the background of the lefthand panel, and selecting the *Add Defined Annotation* menu item.



When the list of defined annotations is displayed, select the *validator* annotation.



After pressing the *Ok* button, the annotation editor will configure the righthand panel with the parameters associated with this annotation.



Destination: {http://www.jboss.org/examples/store}StoreService
Type: service name
Dynamic ReplyTo: ☐

To specify the Endpoint Reference (EPR) for a particular message exchange, enter the EPR into the *Destination* field. The value specified in this field will be dependent upon the technology being validated. For example, if the JBossESB is being monitored, then the value will be a physical address associated with the ESB service endpoint (e.g. jms:queue/esb-quotes). If the technology being validated is a Web Service (or BPEL process), then the field will represent the WSDL service name specified using the QName style (e.g. {namespace}localpart).

The *Type* field is used to define the style of endpoint being validated. In the image above, the endpoint being validated is a Web Service (or BPEL process), and therefore the type is specified as a 'service name'. However if the technology being validated identifies a different endpoint address, for the request and response (as in the case of JBossESB), then the type should be set to 'endpoint address'.

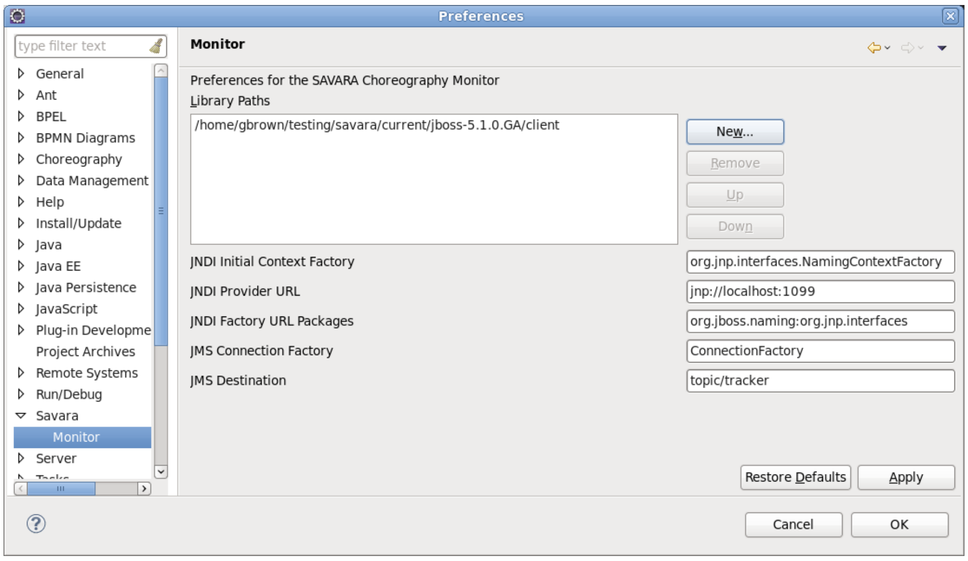
If the exchange is a request, that will result in a response being sent on a dynamically provided "reply-to" destination, then the *Dynamic Reply-To* checkbox should be selected. This situation may occur in the case of validating a JBossESB service, where a well-defined endpoint address has not been defined for the response.

Once the annotation has been defined, then press the *Save* button to save the annotation against the interaction's exchange details.

When all of the relevant 'exchange details' components have been configured with a *validator* annotation, defining the EPR to be validated, then the choreography description file can be copied into the `savara-validator-jboss.sar/models` folder. This will cause the validation mechanism to derive the configuration information from the choreography description model, and begin validating the defined destinations against that choreography description model.

## 3.3. Monitoring the Choreography Description

Once the server environment has been configured, to perform service validation of a set of services against a choreography description, and the server has been started, then the next step is to configure the monitoring tool. This can be achieved by opening the *Window->Preferences->Savara->Monitor* dialog, as shown in the following image.



Monitor

Preferences for the SAVARA Choreography Monitor

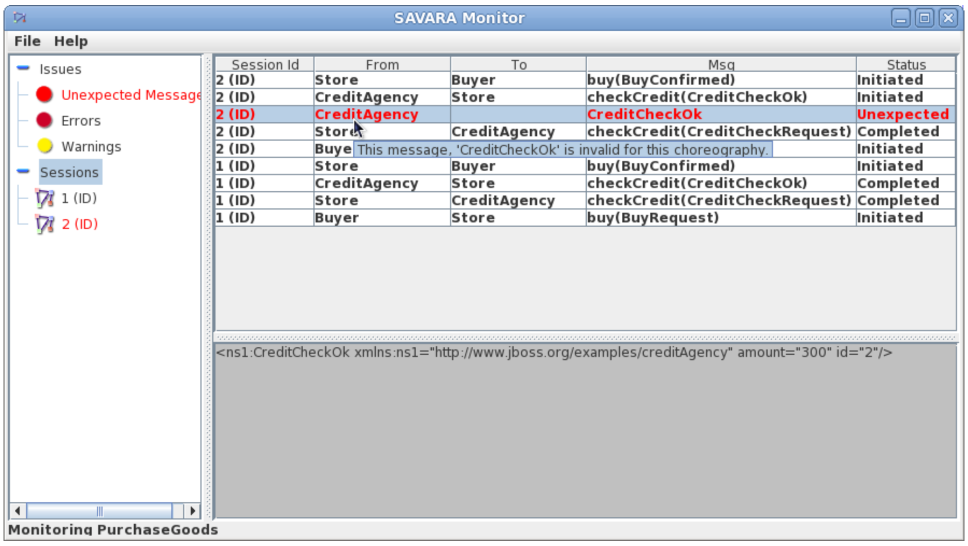
Library Paths

/home/gbrown/testing/savara/current/jboss-5.1.0.GA/client

| | |
|---|---|
| JNDI Initial Context Factory | org.jnp.interfaces.NamingContextFactory |
| JNDI Provider URL | jnp://localhost:1099 |
| JNDI Factory URL Packages | org.jboss.naming:org.jnp.interfaces |
| JMS Connection Factory | ConnectionFactory |
| JMS Destination | topic/tracker |

In general, the default values specified for the JNDI and JMS properties will be fine. The only information that should need to be provided is a path to the JMS and JNDI libraries. When using the JBossAS server, this path can be set to the top level `client` folder.

The next step is to launch the monitoring tool. This is located on the popup menu, for the choreography description (i.e. .cdm) file, by selecting the Savara->Monitor menu item. Once the tool has been launched, it will load the choreography description, subscribe to the relevant event destination, and then indicate via a message in the bottom status line that it is ready to monitor.



| Session Id | From | To | Msg | Status |
|---|---|---|---|---|
| 2 (ID) | Store | Buyer | buy(BuyConfirmed) | Initiated |
| 2 (ID) | CreditAgency | Store | checkCredit(CreditCheckOk) | Initiated |
| 2 (ID) | CreditAgency | | CreditCheckOk | Unexpected |
| 2 (ID) | Store | CreditAgency | checkCredit(CreditCheckRequest) | Completed |
| 2 (ID) | Buye | | | Initiated |
| 1 (ID) | Store | Buyer | buy(BuyConfirmed) | Initiated |
| 1 (ID) | CreditAgency | Store | checkCredit(CreditCheckOk) | Completed |
| 1 (ID) | Store | CreditAgency | checkCredit(CreditCheckRequest) | Completed |
| 1 (ID) | Buyer | Store | buy(BuyRequest) | Initiated |

This message, 'CreditCheckOk' is invalid for this choreography.

```
<ns1:CreditCheckOk xmlns:ns1="http://www.jboss.org/examples/creditAgency" amount="300" id="2"/>
```

Monitoring PurchaseGoods

When the information is received, from the service validators representing the different participants (services), it is correlated to show the global status of the business transaction. The list of correlated interactions is shown in reverse time order in the image.

If any *out of sequence* or other error situations arise, these are displayed in red.

## 3.4. Configuration for Conversation Recording

As well as validating the interactions between a set of services, against a pre-defined choreography description, it is also possible to use the *Service Validators* in a non-validating record mode.

This will be useful in situations where a choreography description does not currently exist, and we wish to use the stream of business events being sent and received by each identified service (or participant type) to gain an understanding of the current business process.

An example of this type of configuration, associated with the TrailBlazer example, is:

```
<validator>
    <service role="LoanBrokerParticipant" validate="false" >
        <output epr="jms:queue/esb-tb-creditAgencyQueue" />
        <input epr="jms:queue/esb-tb-creditAgencyQueue_reply" />
        <output epr="jms:queue/esb-tb-jmsBankRequestQueue" />
        <output epr="jms:queue/esb-tb-fileBankRequestQueue" />
        <input epr="jms:queue/esb-tb-jmsBankResponseQueue" />
        <output epr="jms:queue/esb-tb-customerNotifier" />
        <input epr="jms:queue/esb-tb-fileBankResponseQueue" />
    </service>
    <service role="CreditAgencyParticipant" validate="false" >
        <input epr="jms:queue/esb-tb-creditAgencyQueue" />
        <output epr="jms:queue/esb-tb-creditAgencyQueue_reply" />
    </service>
    <service role="BankParticipant" validate="false" >
        <input epr="jms:queue/esb-tb-jmsBankRequestQueue" />
        <input epr="jms:queue/esb-tb-fileBankRequestQueue" />
        <output epr="jms:queue/esb-tb-jmsBankResponseQueue" />
        <output epr="jms:queue/esb-tb-fileBankResponseQueue" />
    </service>
    <service role="NotifierParticipant" validate="false" >
        <input epr="jms:queue/esb-tb-customerNotifier" />
    </service>
</validator>
```

To define a *Service Validator* in record only mode, the *model* attribute is not specified (because no choreography description exists to be validated against), and the optional *validate* attribute should be set to **false** (by default this attribute is **true**).