Object Life Cycle Explorer for WebSphere Business Modeler

Version 1.1.2

User Guide

September 2008
IBM Zurich Research Laboratory

Contents

1. Overview

Process modeling has proven to be an essential tool for the analysis, design and implementation of applications that automate business process logic. As part of achieving a business goal, a business process typically manipulates several business objects, transforming their states as the process progresses. Understanding the complete state evolution of a single business object is often required for monitoring, governance and compliance purposes, but eliciting such object life cycle information correctly from complex process models can be very challenging. Object Life Cycle Explorer for WebSphere Business Modeler alleviates this problem by allowing you to analyze and design an application from both the process and the object life cycle perspectives.

Object Life Cycle Explorer is a set of plug-ins for WebSphere Business Modeler, a tool that natively supports modeling of business processes, business objects (referred to as business items), resources, etc. WebSphere Business Modeler V6.1 additionally allows you to define a set of possible states for a business item and associate states with object flows in a process model. Object Life Cycle Explorer complements this functionality with the following features:

1. *Object life cycle modeling and visualization*, which allows you to associate state transition diagrams with business items to represent their life cycles. An object life cycle can also be viewed as a state transition table, more suitable for exploring complex life cycles.

2. *Extraction of object life cycles from process models*, which generates object life cycle models capturing state evolution of business items across selected process models. Several options for traversal of subprocesses are provided to produce object life cycles of different granularity.

3. *Checking consistency of process models against given object life cycles*, which identifies activities in process models that induce non-conformant state changes for business items and determines whether some parts of object life cycles are not covered by any process model. Inconsistency resolution support partially automates the process of resolving detected inconsistencies.

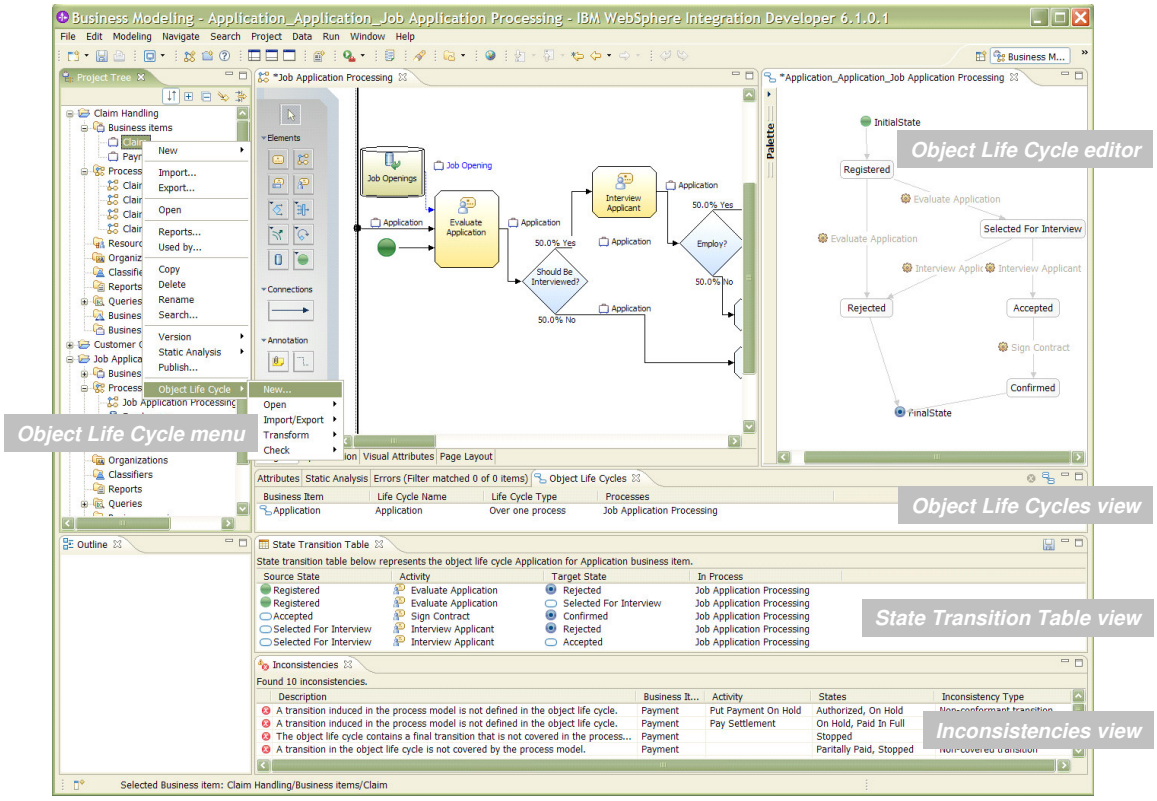
   new in V1.1.0 see Section 6

4. *Generation of a process model from object life cycles*, which synthesizes a process model that manipulates the given business items in accordance with their intended life cycles.

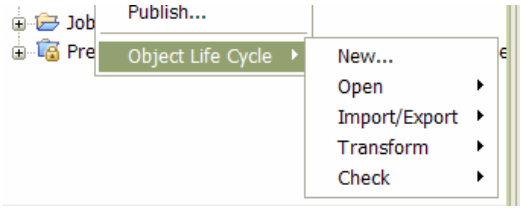
5. *Import and export capabilities,* which facilitate the exchange of object life cycle models between projects and additionally allows users to save object life cycle information in a table format or as an image for use in reports. It is also possible to open an object life cycle model as a Business State Machine in WebSphere Integration Developer for proceeding to build deployable components.

Introduction to the user interface

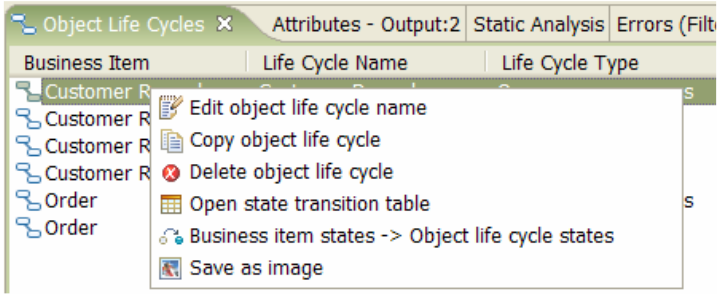
Object Life Cycle Explorer adds a new menu, a state transition diagram editor and several new views to WebSphere Business Modeler:



Options in the Object Life Cycle menu:



Options in the context menu in the Object Life Cycles view:



Integration and Business editions

Object Life Cycle Explorer for WebSphere Business Modeler is available in two editions: *Integration* and *Business*. The Integration edition includes all the outlined features, while the Business edition supports only the extraction of object life cycles from process models and some supporting features. The state transition diagram editor is provided in the Integration edition only. In the Business edition, menu items corresponding to disabled features are grayed-out.

Feature	I	B
Object life cycle modeling and visualization		
• ***Create a new object life cycle*** Object Life Cycle menu: New…	+	-
• ***Open all object life cycles for a project*** Object Life Cycle menu: Open → All Object Life Cycles	+	+
• ***Open all object life cycles for a business item*** Object Life Cycle menu: Open → Object Life Cycles for Business Item	+	+
• ***Rename an object life cycle*** Object Life Cycles view: Edit object life cycle name	+	+
• ***Create a copy of an object life cycle*** Object Life Cycles view: Copy object life cycle	+	-
• ***Delete an object life cycle*** Object Life Cycles view: Delete object life cycle	+	+
• ***Open a life cycle as a state transition table*** Object Life Cycles view: Open state transition table Object Life Cycles view: Double-click on entry (Business edition)	+	+
• ***Open a life cycle as a state transition diagram*** Object Life Cycles view: Double-click on entry (Integration edition)	+	-
• ***Synchronize object life cycle states with business item states*** Object Life Cycles view: Business item states -> Object life cycle states	+	-
Extraction of object life cycles from process models Object Life Cycles view: Transform → Processes -> Object Life Cycles	+	+
Checking consistency of process models against given object life cycles Object Life Cycles view: Check → Process/Object Life Cycle Consistency	+	-
Generation of a process model from object life cycles Object Life Cycles view: Transform → Object Life Cycles -> Processes	+	-
Import and export capabilities		
• ***Table format export*** State Transition Table view: Click on Save button	+	+
• ***Image format export*** Object Life Cycles view: Save as image	+	+
• ***Archive export for project interchange*** Object Life Cycle menu: Import/Export → Export…	+	+
• ***Archive import for project interchange*** Object Life Cycle menu: Import/Export → Import…	+	+

In the remainder of this user guide, parts of the documentation that are only relevant to

- Integration edition will be marked as I
- Business edition will be marked as B

Installations of these two editions have different software prerequisites, as described in the next section.

2. Installing Object Life Cycle Explorer for WebSphere Business Modeler

The download package comprises the following directories and files:

- **license** directory with multi-lingual licensing information
- **samples** directory containing sample models used in tutorials and a **completed** subdirectory with models demonstrating tutorial solutions
- **Object Life Cycle Explorer Business Update Site** directory containing installation files for the Business edition of Object Life Cycle Explorer
- **Object Life Cycle Explorer Integration Update Site** directory containing installation files for the Integration edition of Object Life Cycle Explorer
- this document, **user guide and tutorials.pdf**, which includes an overview of all the tool features, installation instructions, several step-by-step tutorials and an account of current limitations

Software prerequisites

Operating system:

- Windows

Software:

B

I

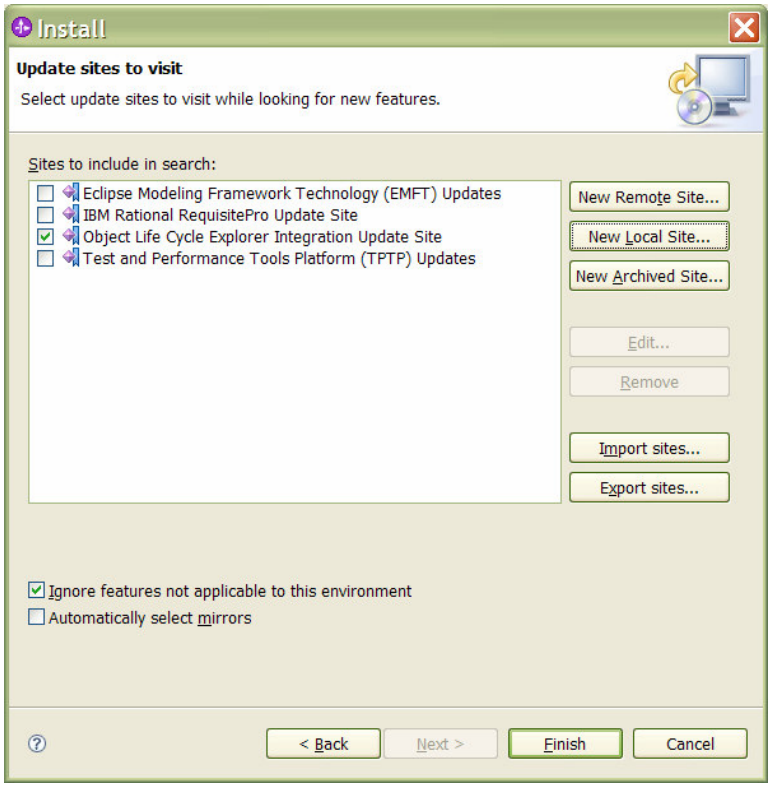
- WebSphere Business Modeler Advanced Version 6.1.X
- WebSphere Business Modeler Advanced Version 6.1.X installed as a set of plug-ins into WebSphere Integration Developer Version 6.1.X

How to install Object Life Cycle Explorer

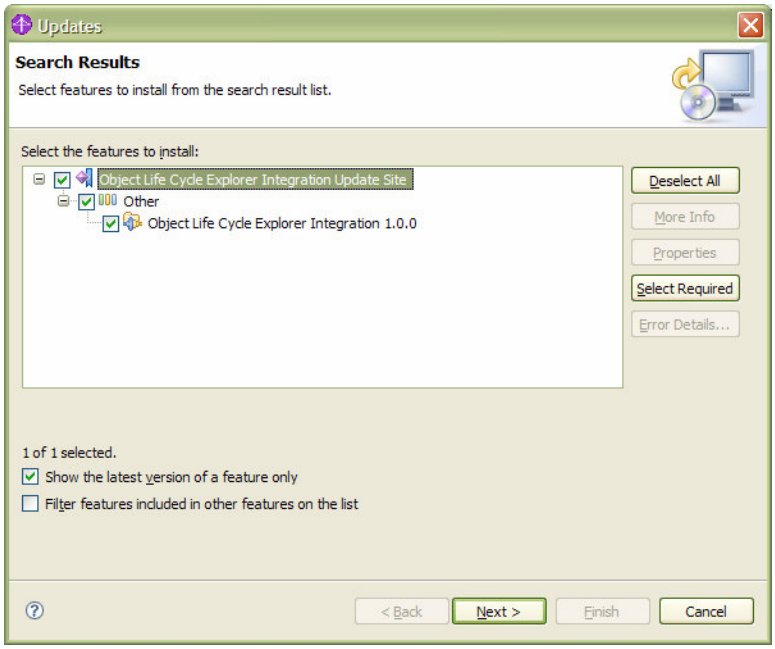
Open WebSphere Business Modeler.

1. Extract the contents of the download package into a **...\Temp** directory.
2. Select **Help → Software Updates → Find and Install…**
3. In the **Install/Update** dialog box, select **Search for new features to install** and click on **Next**.
4. In the **Install** dialog box, click on **New Local Site**.
5. In the file dialog box, navigate to the **...\Temp\Object Life Cycle Explorer Integration Update Site** or **...\Temp\Object Life Cycle Explorer Business Update Site** directory, depending which edition you want to install. Click on **OK**.
6. Click on **OK** in the **Edit Local Site** dialog box.

7. A new site for Object Life Cycle Explorer is added to the Sites list. Make sure this site is selected and click on **Finish**.



8. In the **Updates** dialog box, select all features for this site and click on **Next**.



9. Read the license information and, if you agree, select **I accept the terms in the license agreement.** Click on **Next.**

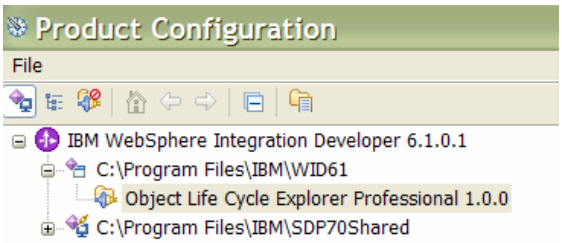
10. Click on **Finish** on the next page to confirm the **Installation Location**. Because the feature is not digitally signed, a warning might be displayed. Click on **Install** if you want to continue.
11. In the final dialog box, click on **Yes** to restart WebSphere Business Modeler.

Attention: It is not recommended to have the Integration and the Business editions installed on the same machine at the same time, even if you have two installations of WebSphere Business Modeler (one standalone and the other in WebSphere Integration Developer). Make sure that you uninstall one edition before installing the other.

How to uninstall Object Life Cycle Explorer

Open WebSphere Business Modeler.

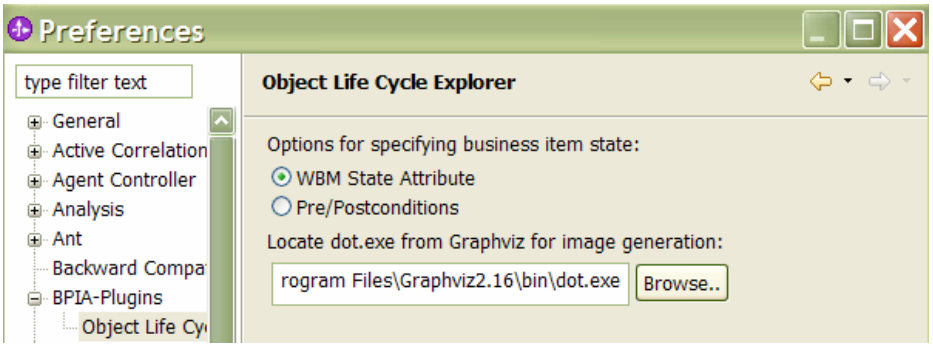
1. Select **Help → Software Updates → Manage Configuration**.
2. Expand the Feature Tree to see installed features.



3. Right-click on Object Life Cycle Explorer feature and select **Uninstall**.

Configuring image generation using dot from Graphviz

Object Life Cycle Explorer allows you to export an object life cycle as an image. This is especially valuable for the Business edition, which does not include a state transition diagram editor. To enable the image export, you need to install the Graphviz software, available here: http://www.graphviz.org. Once Graphviz is installed, go to **Window → Preferences** in WebSphere Business Modeler. Open the **BPIA-Plugins → Object Life Cycle Explorer** preference page. Click on **Browse** and navigate to the location of the **dot.exe** in the Graphviz install directory. Click on **OK**.



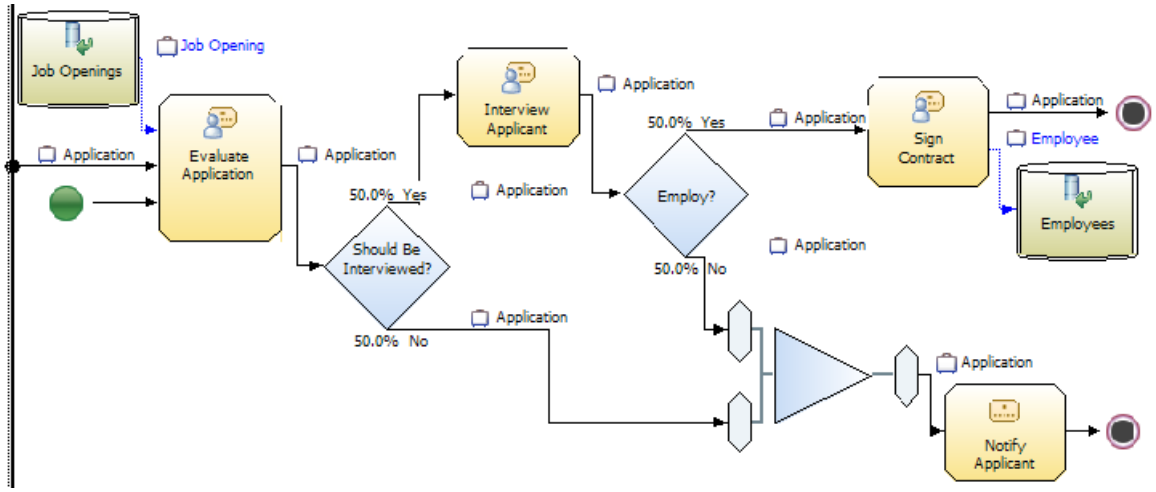
3. Getting Started

This user guide assumes that you are already familiar with the most common features of WebSphere Business Modeler. This short exercise explains how in just a few simple steps you can start exploring the life cycles of your business items.

In this exercise you will learn how to:

- add states to business items (see also WebSphere Business Modeler Help),
- specify business item states in a process model (see also WebSphere Business Modeler Help),
- and extract object life cycles from a process model.

Import the **Job Application Processing** project (**Job Application Processing.mar** file in the **samples** directory) and open the **Job Application Processing** process model. In the Basic mode, this process model looks as follows:

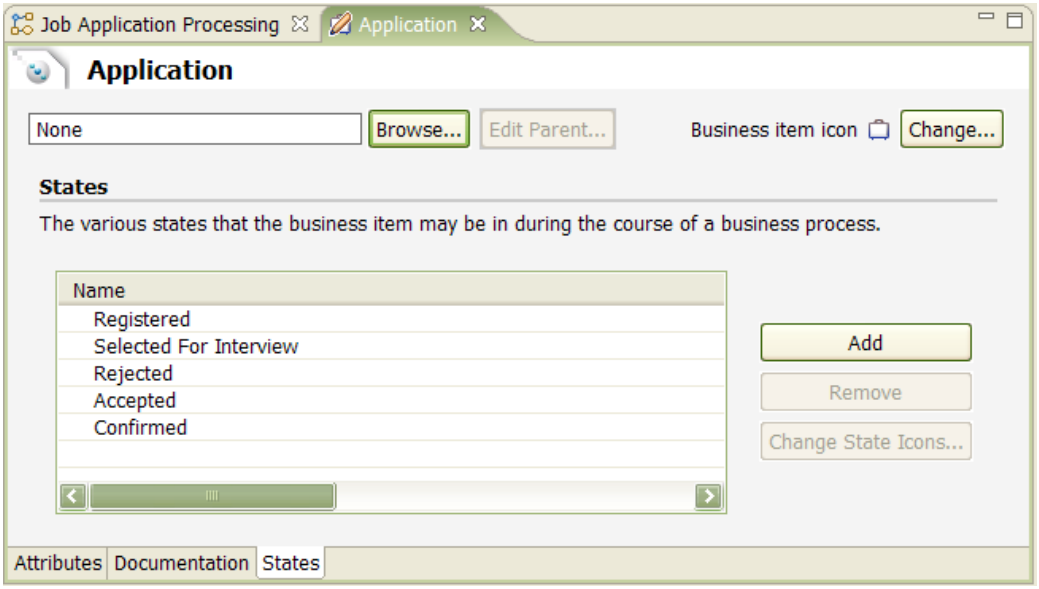


This process model demonstrates simple handling of job applications. An incoming **Application** is first evaluated against the **Job Opening** requirements. Selected applicants are interviewed and those that perform well in the interview sign a contract and become an **Employee** of the company. In case an **Application** is rejected before or after the interview, the applicant is notified.

Processing one application may last several weeks or even months and thus it is important to keep track of the status of each application. To do this, you need to specify how the activities in the process model change the state of the **Application** business item. Possible states of the **Application** business item need to be defined first.

Adding states to a business item

1. Open the business item editor for the **Application** business item.
2. Select the **States** tab.
3. Repeatedly add states by clicking on the **Add** button and renaming the newly added states to obtain states **Registered**, **Selected For Interview**, **Rejected**, **Accepted** and **Confirmed**.
4. Save the **Application** business item.

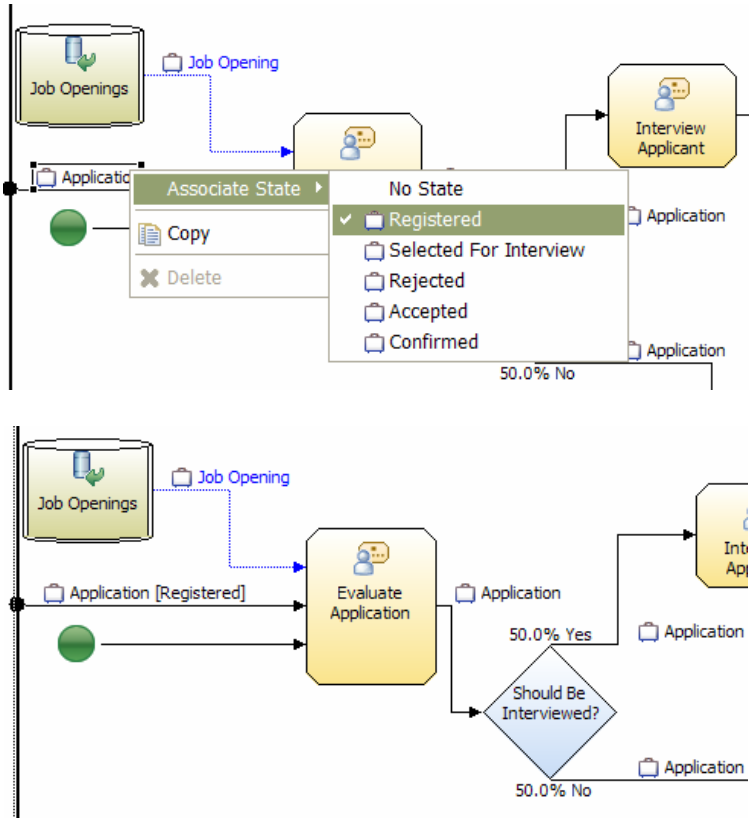


Now you can use these states in the process model. Go back to the **Job Application Processing** process model editor.

Specifying business item states in a process model

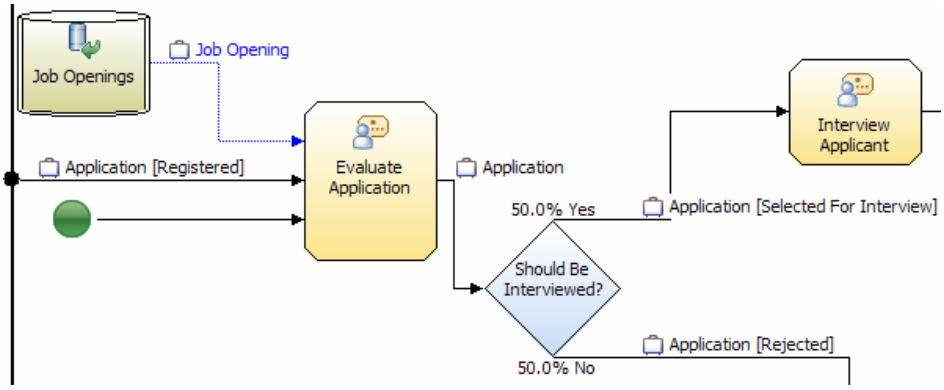
A business item is generally either created by an activity inside a given process model or passed to the process model via an input parameter. You should begin by specifying the states of business items at these points in the process model. In the **Job Application Processing** example, the **Application** business item is received via an input parameter. By the time the **Application** is received by this process, it has already been registered in the system and therefore should be assigned the state **Registered**.

1. Right-click on the label of the object flow (an object flow is a connection associated with a business item) connecting the process boundary and the **Evaluate Application** task.
2. Select **Associate State → Registered**.

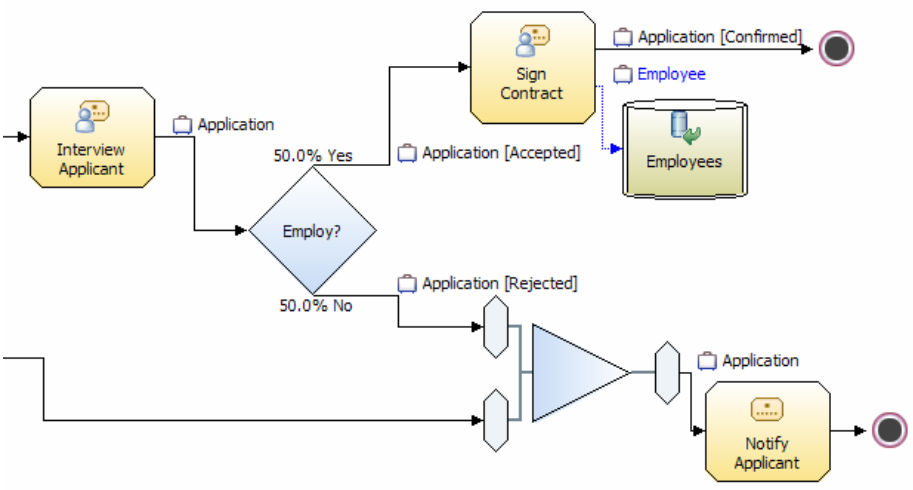


Once you have specified the states in which business items are created and received, proceed to identify all the activities in the process that change the state of business items. In this example, after the **Evaluate Application** task completes, the **Application** state is changed to either **Selected For Interview** or **Rejected**. Currently in WebSphere Business Modeler, you cannot directly assign several alternative states to a single object output of an activity. Instead, the activity must be followed by a decision that splits the process flow according to the state of the business item.

In this example, the decision **Should Be Interviewed?** passes the **Applications** in state **Selected For Interview** to the **Interview Applicant** task and those in state **Rejected** to the **Notify Applicant** task. Specify the states of the edge going out of the decision to reflect this, to obtain the following result:



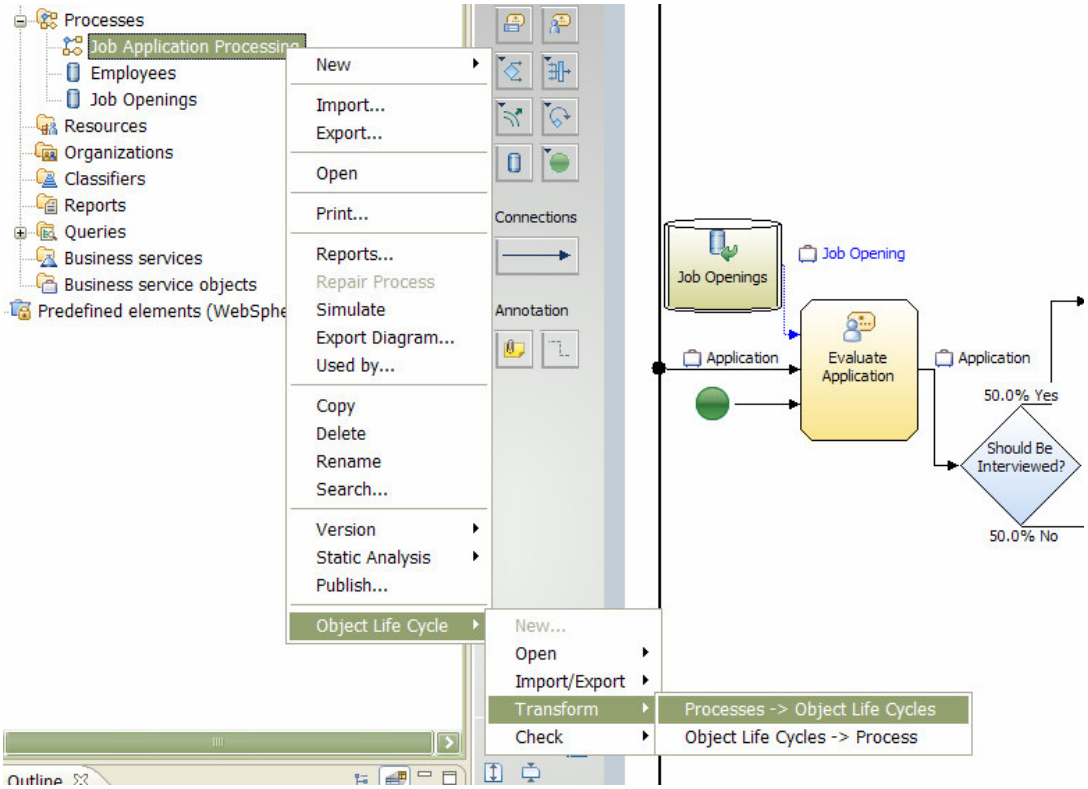
Similarly, specify the states for the object flows going out of the **Employ?** decision to reflect that the **Interview Applicant** task updates the state of the **Application** to either **Accepted** or **Rejected**. Finally, specify that the **Sign Contract** task changes the **Application** state to **Confirmed**. Save the process model.



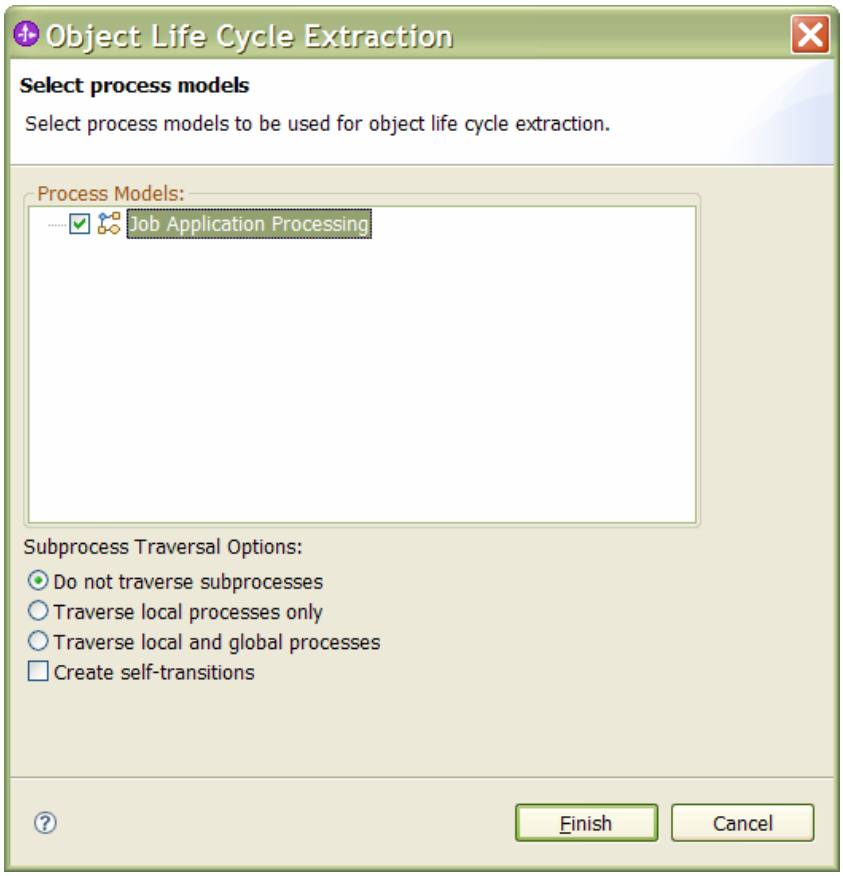
Extracting an object life cycle from a process model

You can now generate an object life cycle model for the **Application** business item.

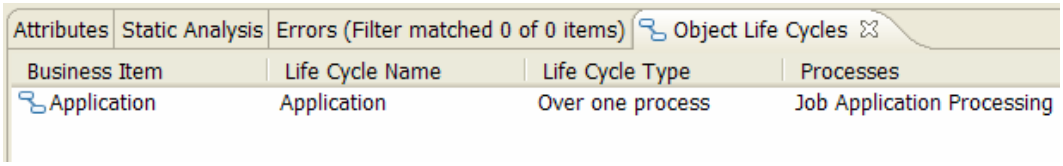
1. Right-click on the **Job Application Processing** process in the Project Tree and select **Object Life Cycle → Transform → Processes -> Object Life Cycles**.



2. Keep the default options in the **Object Life Cycle Extraction** wizard and click on **Finish** button.

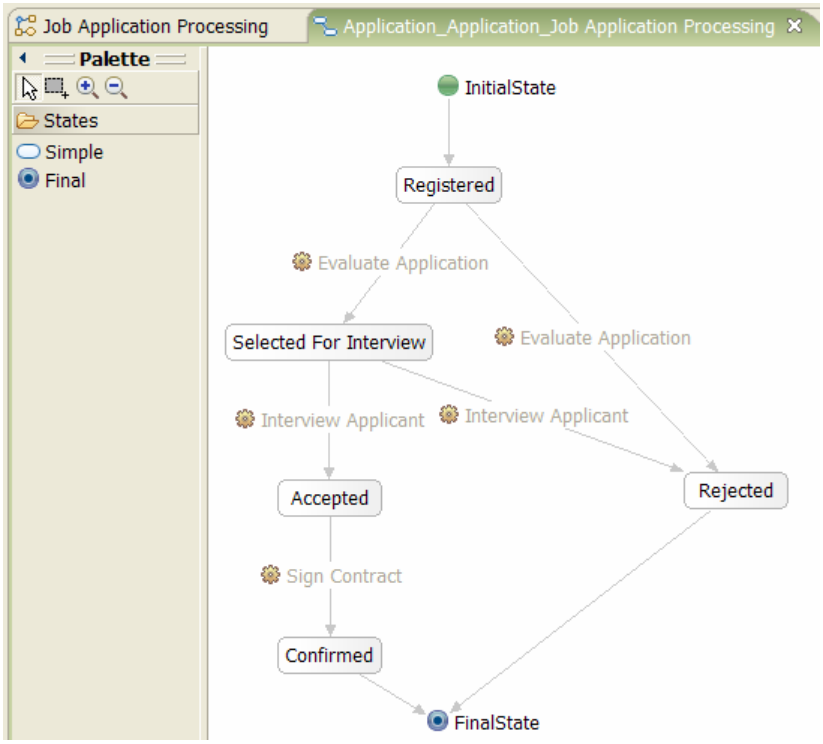


3. The generated object life cycle for the **Application** business item is shown in the opened **Object Life Cycles** view.



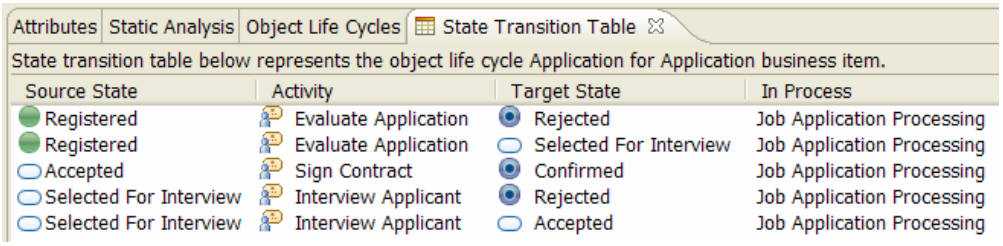
I

Double-click the only entry in this view to open the extracted object life cycle in the **Object Life Cycle** editor. You may need to move around some of the states and transitions in the state transition diagram to get a good layout. Each transition in this object life cycle is labeled with an activity name from the **Job Application Processing** process model, except for some transitions from the initial and to the final states.



B

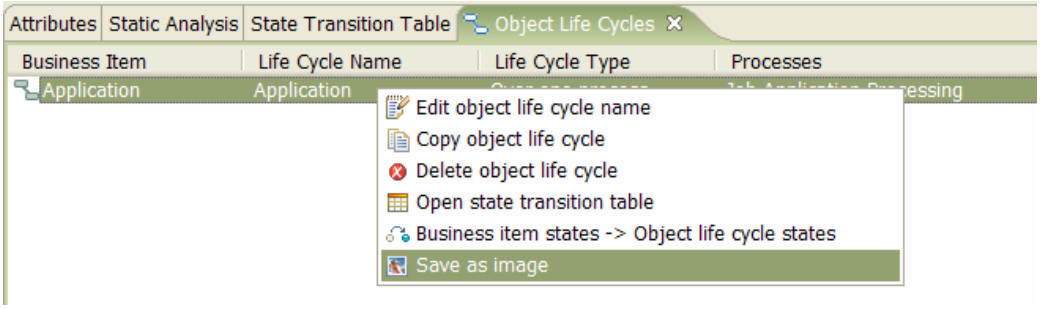
Double-click on the only entry in the Object Life Cycles view to open the extracted object life cycle in the **State Transition Table** view.



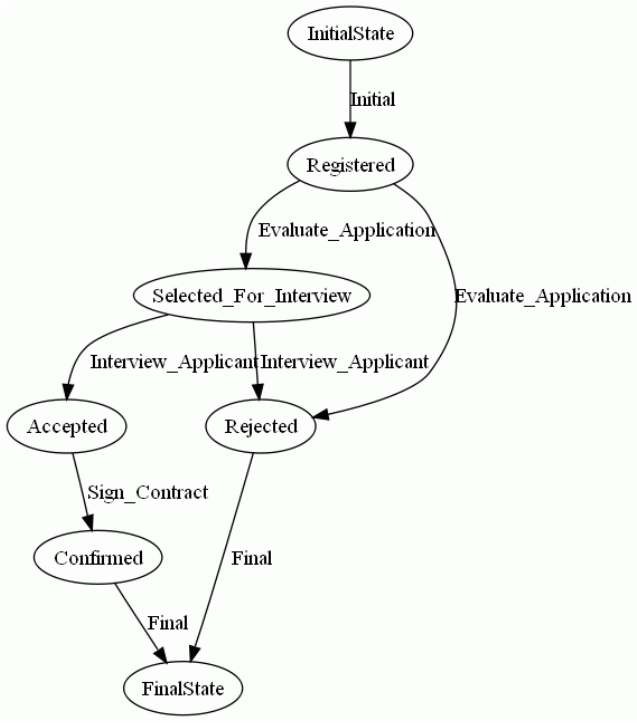
Each row in the State Transition Table view corresponds to an activity in the **Job Application Processing** process model that changes the state of the **Application** business item. Each such activity represents a state transition in this object life cycle. The source and target states of the state transitions are shown in the first and third columns, respectively. Green circles mark those states in which the business item is either created or passed to the process model via an input parameter. Blue circles with a dark blue dot in the middle mark the last states that the business item reaches in the process model. The fourth column gives the name of the process model from which the transition was

extracted. The rows can be sorted by the contents of each column by clicking on the corresponding column label.

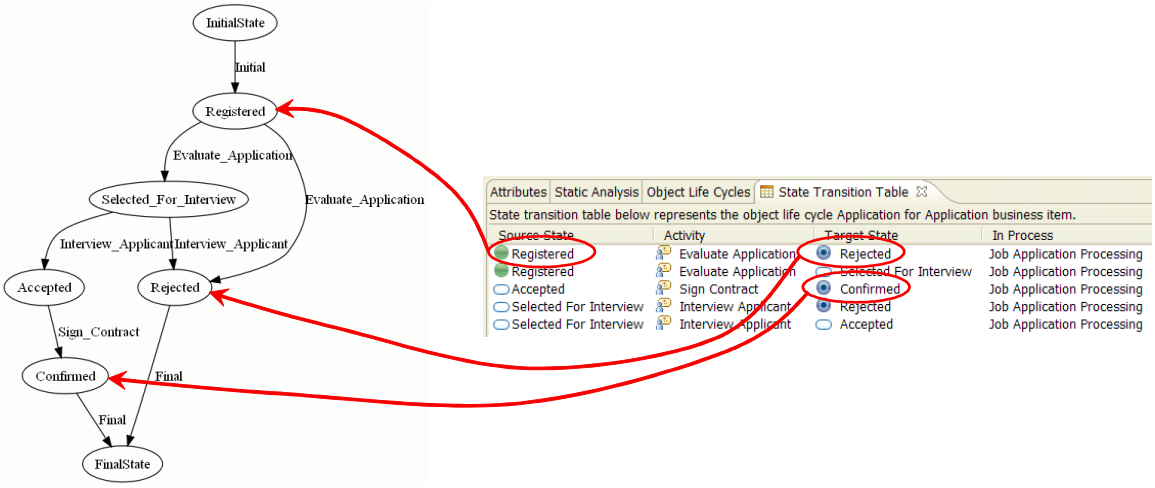
Provided that you have **dot** from **Graphviz** installed (see installation instructions in Section 2), you can additionally generate an image containing a state transition diagram to help you understand an extracted object life cycle. Right-click on the entry for the **Application** object life cycle in the Object Life Cycles view and select **Save as image**.



In the file dialog box, navigate to the location where you want to save the image file and click on **Save** (if you do not have Graphviz installed or it is not configured correctly, an error message is displayed). You can then open the generated image from the file system. This state transition diagram comprises the same states and transitions as the ones shown in the Object Life Cycle editor in the Integration edition of Object Life Cycle Explorer (see previous page). Note that all spaces in state and activity names are replaced with underscores for compatibility with Graphviz. **Initial** and **Final** are used as default names for the transitions that are not associated with any activity. Running **Save as image** several times on the same object life cycle may produce images with different layout.



After examining this state transition diagram, you should see that each state marked with a green circle in the State Transition Table view corresponds to a state that has a transition from the initial state in the state transition diagram. Each state marked with a blue circle with a dot corresponds to a state that has a transition to the final state.



The extracted object life cycle shows the complete state evolution of the **Application** business item. As opposed to the original process model, which emphasizes the activities and deals with several business items, the object life cycle view is concerned with one business item only and places the main focus on its states and state transitions.

In this particular example, it was not difficult to understand the complete state evolution of the **Application** business item by following the flows in the process model. However, this becomes more challenging when you deal with a complex process model or when one business item is manipulated across several process models. Proceed to **Tutorial 1** to learn more about this.

4. Tutorial 1: Eliciting End-to-end Object Life Cycles

This tutorial assumes that you have completed the **Getting Started** exercise and know how to specify states of business items in a process model and how to extract object life cycles from a process model.

This tutorial will show you how to:

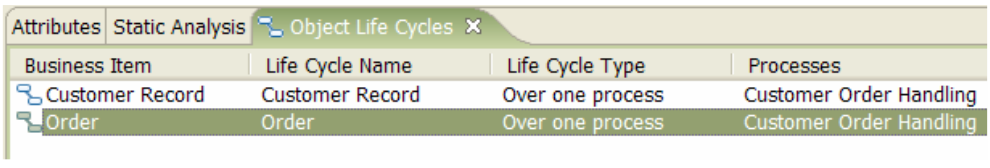
- use object life cycle extraction to ensure a correct specification of business item states in a process model,
- extract object life cycles of different granularity,
- extract object life cycles from several process models and explore them as state transition tables,
- export object life cycles in different formats.

Import the **Customer Order Processing** project (**Customer Order Processing.mar** in the **samples** directory) and open the **Customer Order Handling** process model. In the modeled process, an incoming customer call about an order is handled. If the customer has not ordered with the company before, a new record is created and then a new order is initiated. Otherwise, either an existing order is updated or a new order for an existing customer is created. The customer details for the order are put through a credit check and if successful, the order is sent to shipping. Otherwise, the order is rejected.

States of the **Customer Record** and **Order** are already specified in the process model to reflect the progress of the process. In the following, you will examine the generated object life cycles for these two business items to ensure that their states are correctly specified.

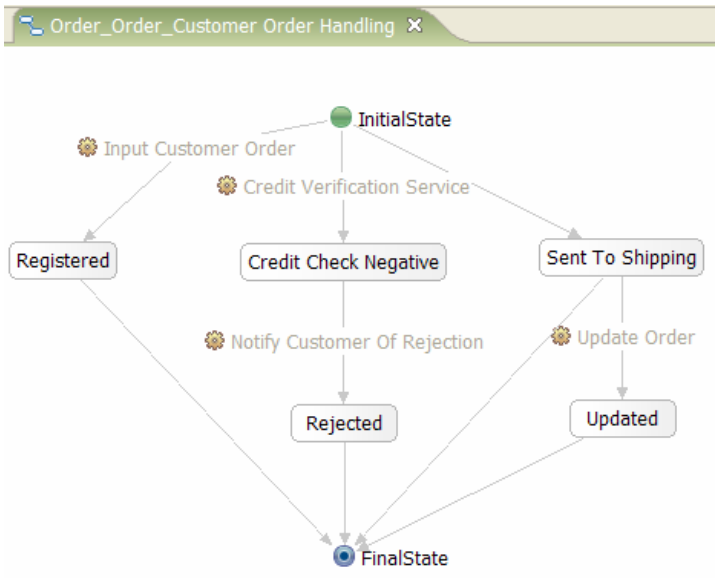
Ensuring a correct state specification using object life cycles

Run the object life cycle extraction from the **Customer Order Handling** process model using the default options. Two object life cycles are generated: one for the **Customer Record** business item and one for the **Order** business item, shown as the two entries in the Object Life Cycles view:

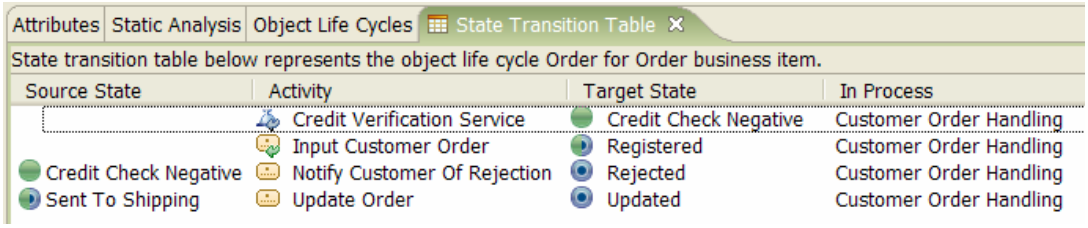


Double-click on the **Order** entry to open the object life cycle for the **Order** business item.

I



B



You can see that states **Sent To Shipping** and **Registered** are marked with a hybrid icon, which indicates that for each of these states there is a transition from the initial state and a transition to the final state. If you prefer a graphical visualization of object life cycles, generate an image containing a state transition diagram using the **Save as image** feature.

Does the extracted object life cycle meet your expectations with respect to the state evolution of the **Order** business item? To answer this question, you can follow the following **5-step Object Life Cycle Validation** method:

1. *Examine each initial transition for validity.* In a state transition diagram (Object Life Cycle editor or generated image), each transition from the initial state is an *initial transition*. In a state transition table, green circles mark those states that have incoming initial transitions. An initial transition is valid if the business item should be either created in the target state of this transition inside the process model or it should be passed to the process model in this state via an input parameter.

    - Is the initial transition to state **Registered** valid? Yes, since an **Order** should indeed be created in state **Registered** in this process model.

- Is the initial transition to state **Credit Check Negative** valid? No, an **Order** should be neither created nor passed to this process in this state.
- Is the initial transition to state **Sent To Shipping** valid? No, an **Order** should be neither created nor passed to this process in this state.

2. *Examine each final transition for validity.* In a state transition diagram, each transition to the final state is a *final transition*. In a state transition table, blue circles with a dot mark those states that have outgoing final transitions. A final transition is valid if the source state of this transition should indeed be the last state that the business item reaches in this process model.

    - Is the final transition from state **Registered** valid? No, since an **Order** should be processed further than the state **Registered** in this process.
    - Is the final transition from state **Rejected** valid? Yes.
    - Is the final transition from state **Updated** valid? No, since an **Order** should once again go through a credit check once it has been **Updated**.

3. *Examine each intermediate transition for validity.* A transition that is not connected to an initial or a final state in a state transition diagram is an intermediate transition. In a state transition table, intermediate transitions correspond to rows that contain both source and target states. Check that the activity associated with each intermediate transition should indeed change the source state of the business item to the target state.

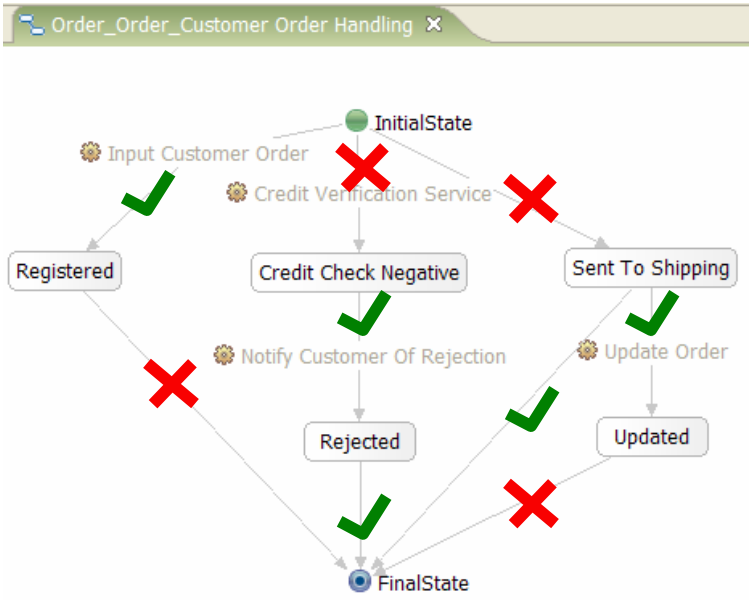
    - Should the **Notify Customer Of Rejection** activity induce a transition from state **Credit Check Negative** to state **Rejected**? Yes.
    - Should the **Update Order** activity induce a transition from state **Sent To Shipping** to state **Updated**? Yes.

4. *Check transition splits for validity.* In a state transition diagram, a transition split corresponds to a state with several outgoing transitions, where each transition is labeled with an activity name. In a state transition table, a transition split occurs when several rows have the same source state, but different activities. A transition split thus means that more than one activity can change the state of the business item from the same source state. Ensure that each such choice is indeed intended.

    - There are no transition splits in this example.

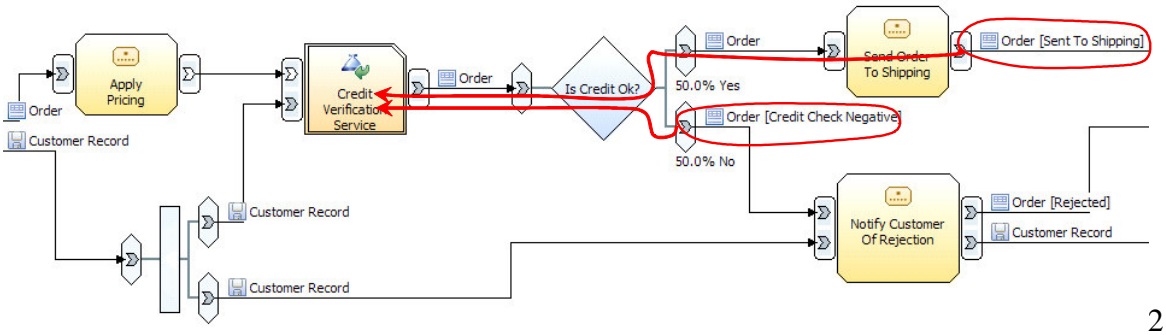
5. *Find missing transitions.* Should the activities in the process model induce other transitions on the business item? Are there activities missing in the process model that should induce transitions on the business item? These questions may be difficult to answer if several invalidities have been identified in steps 1-3. You may need to address those invalidities first and then come back to this step in a later iteration.

At the end of the 5-step Object Life Cycle Validation, each transition is identified as either valid or invalid. For this example, the following results were obtained:

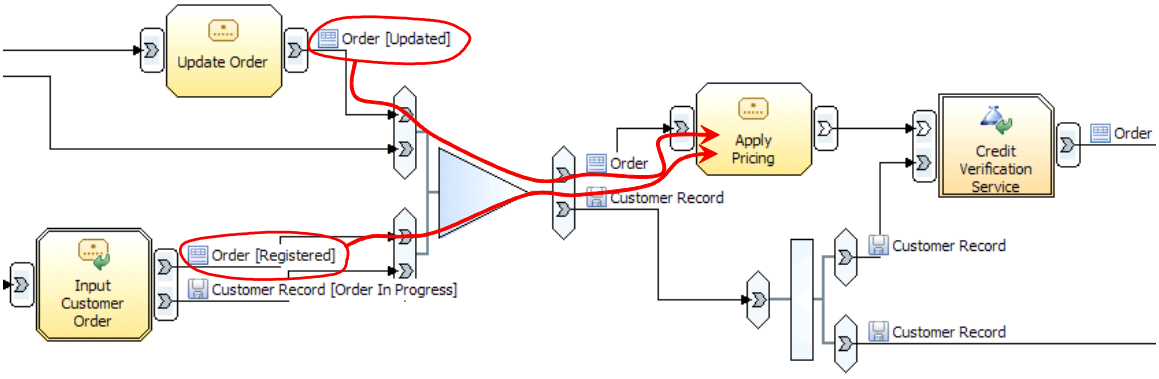


You now need to examine the process model and determine why invalid transitions were generated. Given an invalid initial transition, locate its target state in the process model and follow the object flow of this business item upstream searching for an activity that creates the business item or an input parameter to the process model. An activity that does not have any inputs of a particular business item, but has outputs of this business item is interpreted as a creation activity. Switching to **Intermediate** or **Advanced** mode makes it easier to distinguish the different types of process flows (control flow / object flow). Ensure that the creation activity and input parameter are assigned correct states for this business item. Similarly, given an invalid final transition, locate its source state in the process model and follow the object flow downstream searching for an activity that does not pass on this business item or an output parameter of the process model. Ensure that these are assigned correct states.

Locate **Credit Check Negative** and **Sent To Shipping** states in the process model and follow the object flows of business item **Order** upstream. As shown below, this should lead you to the **Credit Verification Service** activity, which has no inputs of business item **Order**. For this reason, **Credit Check Negative** and **Sent To Shipping** are identified as targets of initial transitions in the extracted object life cycle for **Order**. Does the **Credit Verification Service** really create a new **Order**? No, it should rather receive an existing



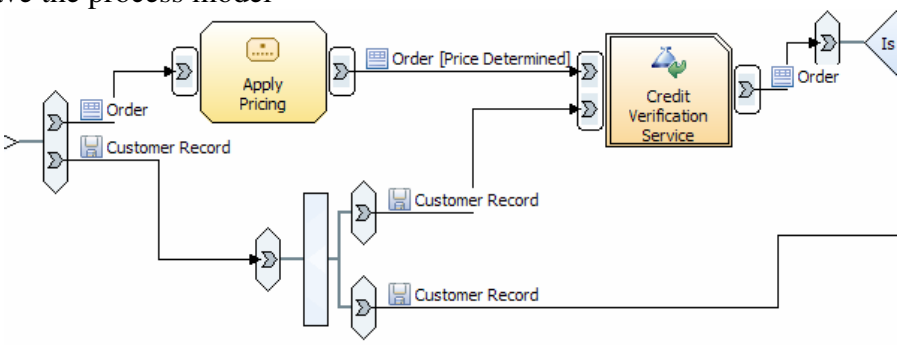
Before making any changes to the process model, examine also the reasons for invalid final transitions. Locate the states **Registered** and **Updated** in the process model and follow the object flows of business item **Order** downstream. As shown below, this leads to the **Apply Pricing** activity, which takes an **Order** as an input, but does not pass it on further in the process. Does this activity make any changes to an **Order**? Does it change the state of an **Order**? Yes, in fact determining the price of an **Order** marks a significant point in the life cycle of this business item, hence **Apply Pricing** should have **Order** as an output with an updated state.



In this example, incorrectly modeled object flow was the cause for invalid initial and final transitions in the extracted object life cycle. If the same business item is used by several activities in a process model, the business item needs to be routed between these activities using object flows.

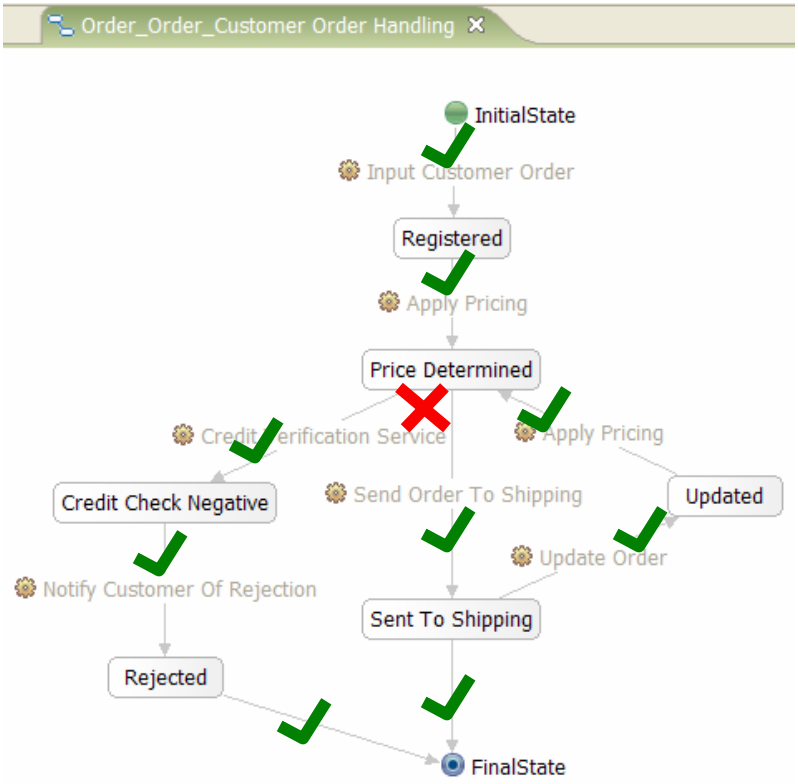
Adjust the process model as follows to address the identified problem:

- Remove the control flow between **Apply Pricing** and **Credit Verification Service** together with the corresponding control input and output
- Add an output of business item **Order** to the **Apply Pricing** activity
- Add an input of business item **Order** to the **Credit Verification Service** activity (right-click on this activity and select **Launch Global Service Editor** to make and save the changes and then right-click on this activity in the **Customer Order Handling** process model and select **Update Global Element**)
- Connect the newly created output and input with an object flow
- Create a new **Price Determined** state for the **Order** business item
- Assign the **Price Determined** state to the output of the **Apply Pricing** activity (right-click on the output to assign the state and not on the object flow)
- Save the process model

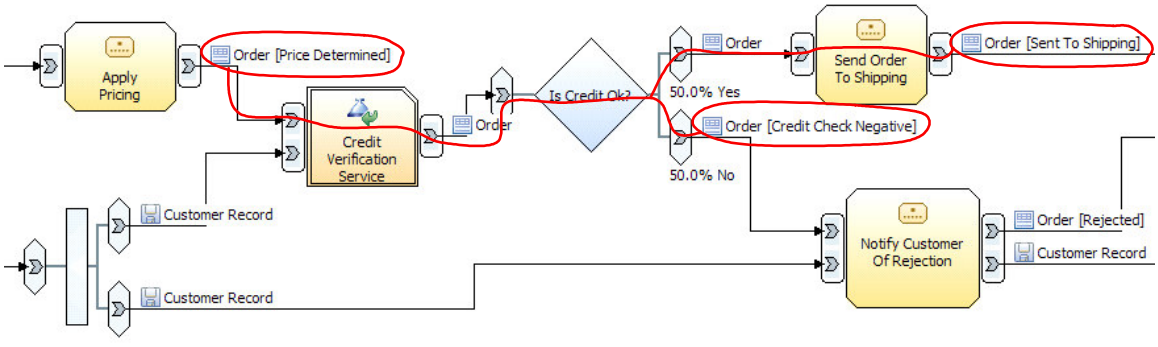


Extract the object life cycles from the adapted **Customer Order Handling** process model to study the effect of the changes. Open the extracted **Order** object life cycle. Repeat the 5-step Object Life Cycle Validation method to check the extracted life cycle.

It is evident that the invalid initial and final transitions encountered previously have been resolved. Throughout steps 1-3, you should not encounter any problems. In step 4 however, you should find an invalid transition split from state **Price Determined**. According to this object life cycle, once an **Order** is in state **Price Determined**, either the **Credit Verification Service** activity is performed to always change the state of **Order** to **Credit Check Negative**, or the **Send Order To Shipping** activity is carried out. However, the requirement is that before any **Order** is sent to shipping, an approval should be received from the **Credit Verification Service**.



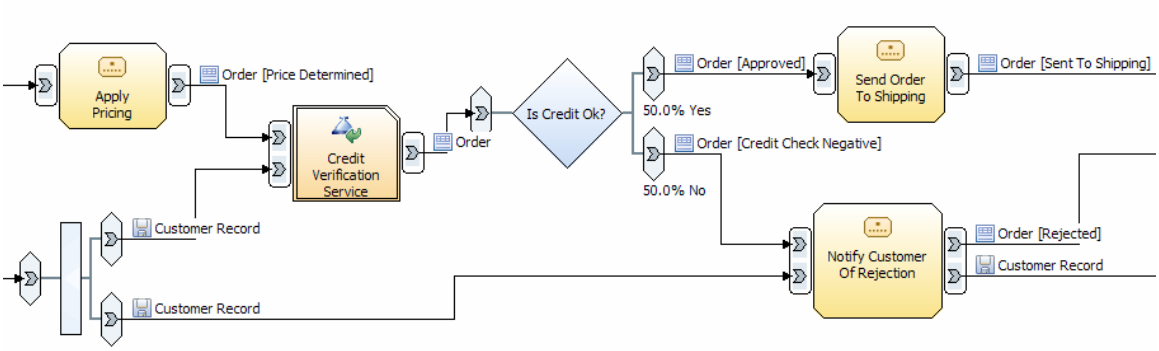
Examine the process model to track down the source of this problem. Locate the source and target states of the transition split, here these are **Price Determined**, **Credit Check Negative** and **Sent To Shipping**, and trace the object flows between these states. Check whether intermediate states should be specified on these object flows or the object flows themselves should be adapted.



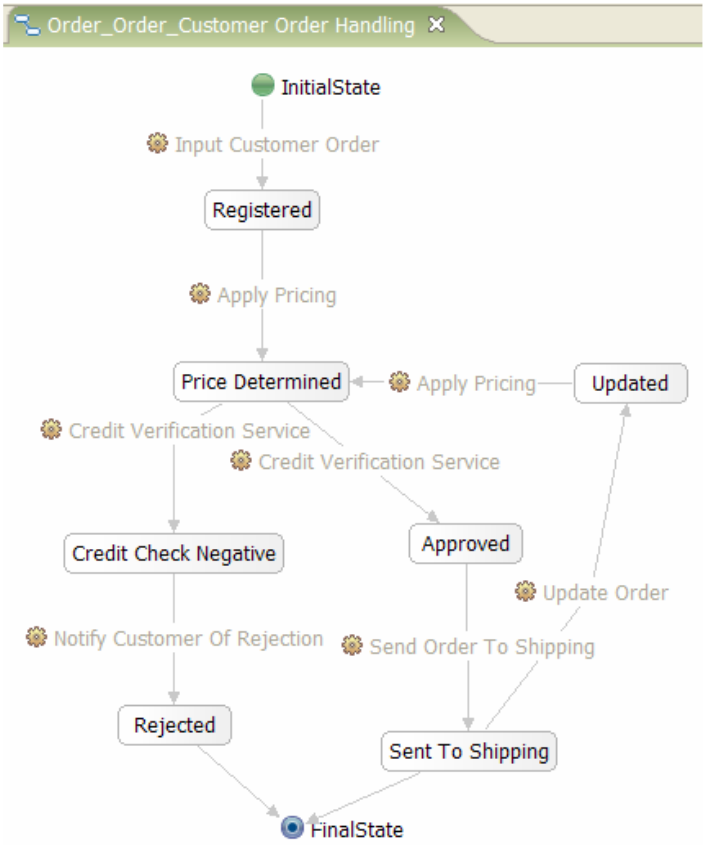
In this example, there is an intermediate state missing on one of the object flows. The upper branch of the **Is Credit Ok?** decision should be associated with a state that indicates an approval of the credit verification.

Correct the process model as follows:

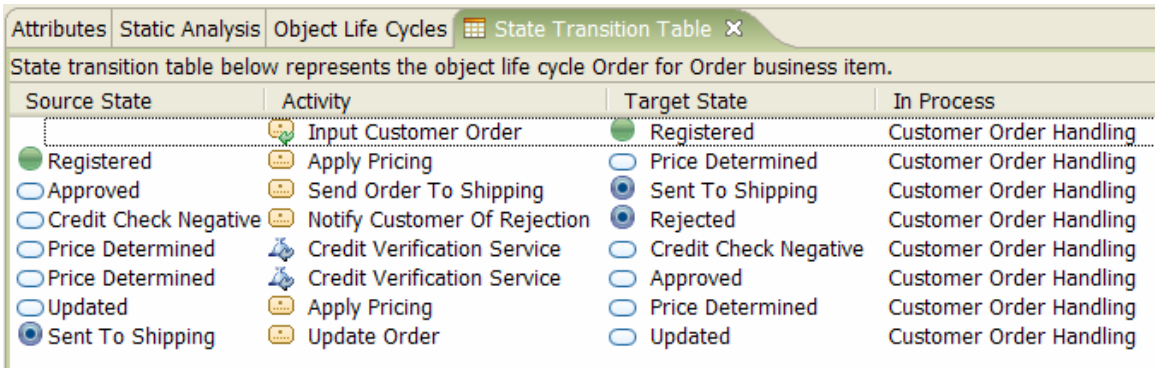
- Add a new **Approved** state to the **Order** business item
- Associate the object flow connecting the **Yes** branch of the **Is Credit Ok?** decision and the **Sent Order To Shipping** activity with the **Approved** state



Extract object life cycles once again to obtain an **Order** life cycle, which should finally meet your expectations.



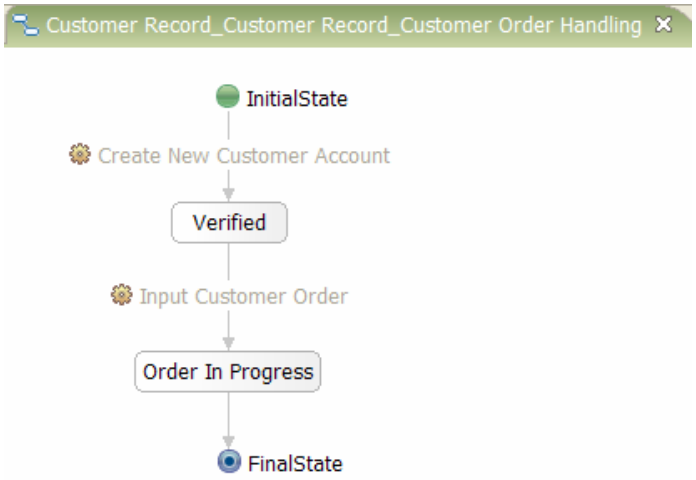
B



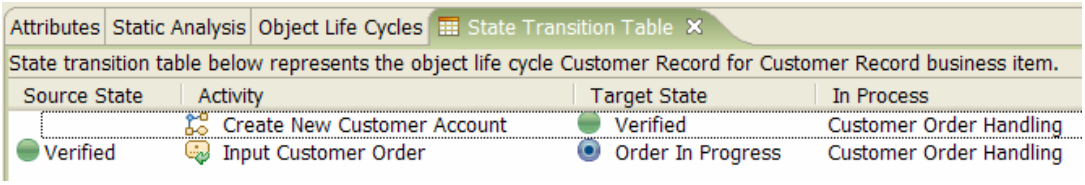
Generating object life cycles with subprocess traversal

Apart from the **Order** business item, which has been your focus up till now, the state of the **Customer Record** business item is also changed in the **Customer Order Handling** process model. Open the object life cycle extracted for **Customer Record**:

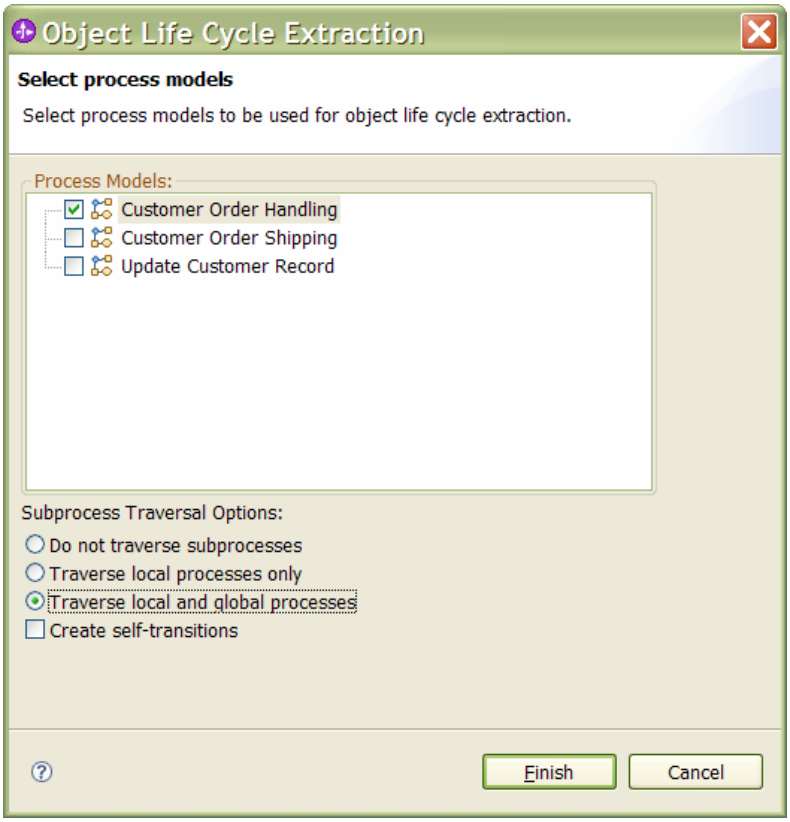
I



B

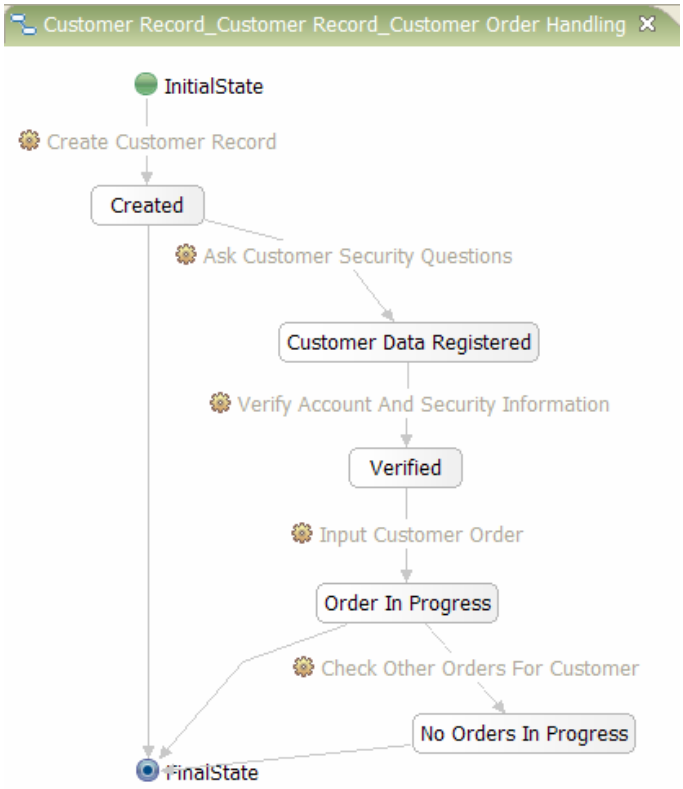


This object life cycle evidently does not show much detail. In order to collect the state information also from all the local and global subprocesses of a process model, you need to ensure that the object life cycle extraction is invoked with the correct subprocess traversal option. Invoke the object life cycle extraction again with the **Subprocess Traversal Option** set to **Traverse local and global processes**, to make sure that all the state evolution information is extracted.

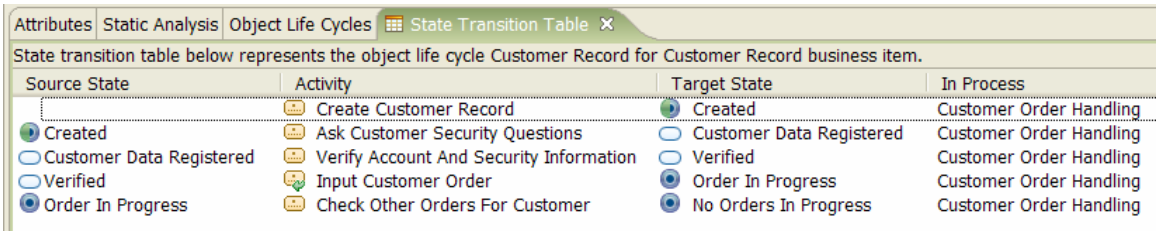


This time the extraction traverses the **Create New Customer Account** local process and the **Update Customer Record** global process to generate a more complete object life cycle for **Customer Record**:

I

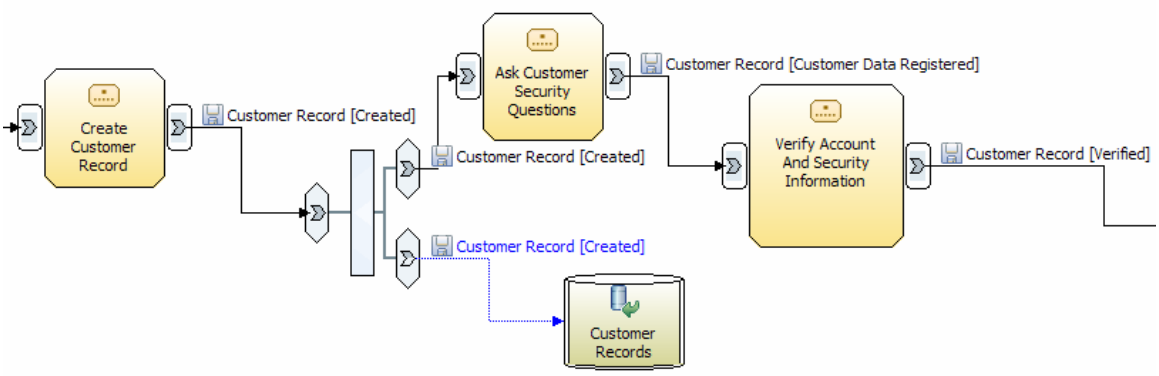


B



Note that **Created** is marked with a hybrid initial/final icon, meaning that it has an incoming transition from the initial state and an outgoing transition to the final state.

If you now examine this object life cycle using the 5-step Object Life Cycle Validation method, you may identify an invalid final transition from state **Created**. In the **Create New Customer Account** subprocess, a new **Customer Record** in state **Created** is produced by the **Create Customer Record** task. The **Customer Record** is then placed into the **Customer Records** repository and also routed to the **Ask Customer Security Questions** task. The object life cycle extraction identifies the repository as a possible end point of the **Customer Record** life cycle and hence generates a transition from **Created** to the final state.

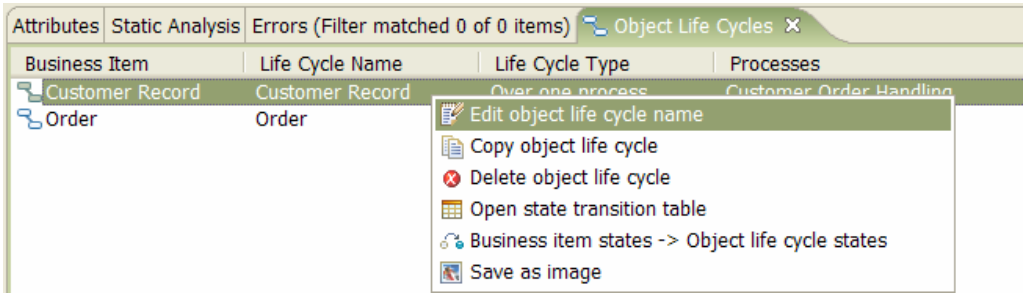


Note that in WebSphere Business Modeler, passing a business item through a fork can be interpreted either as a copy of a reference to the business item instance or an actual copy of the business item instance. In the latter case, there may exist several instances of the same business item during the process execution. The extracted object life cycle captures the amalgamation of state evolution for all such instances.

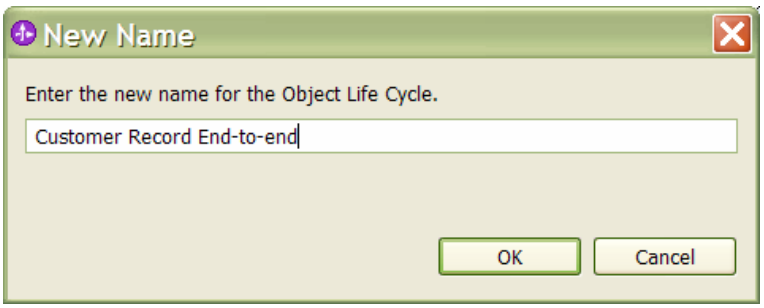
I

You may remove the transition from state **Created** to the final state in the Object Life Cycle editor and save the object life cycle.

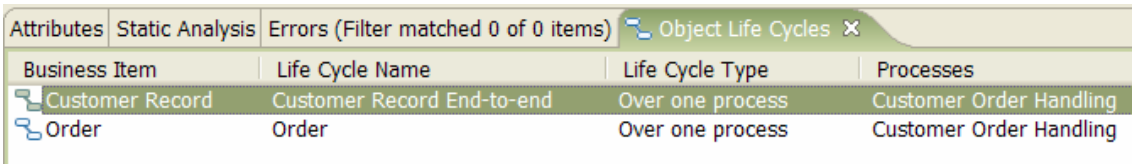
Once you are satisfied with an object life cycle that has been extracted from a process model, it makes sense to rename it, to avoid it being overwritten by later runs of the object life cycle extraction. Right-click on the **Customer Record** entry in the Object Life Cycles view and select **Edit object life cycle name**.



Type the new name in the **New Name** dialog box and click on **OK**.

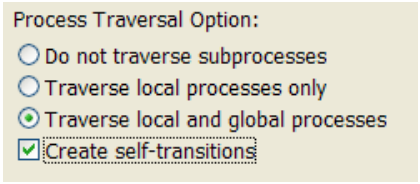


The name of the object life cycle changes, as shown in the Object Life Cycles view.



You can also try to run the object life cycle extraction using the **Traverse local processes only** option and examine the difference to the life cycle previously extracted with full traversal.

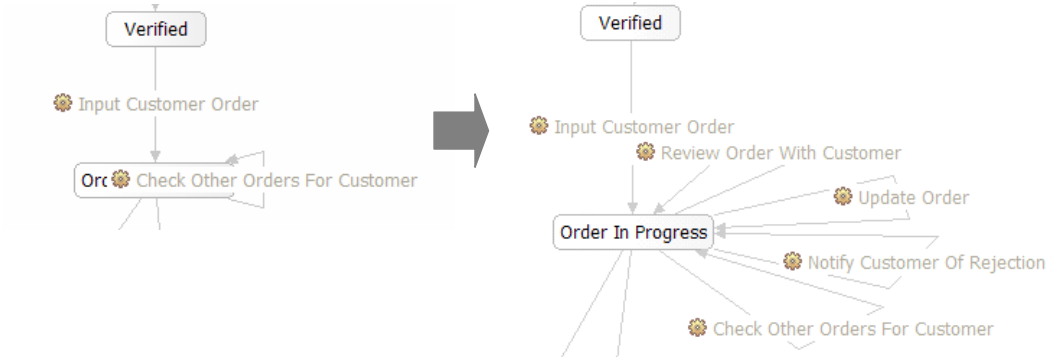
By default, the object life cycle extraction does not produce self-transitions (transitions that have the same source and target states) in the object life cycles. Activities that do not change the state of a business item give rise to self-transitions. To see the self-transitions, run the object life cycle extraction from the **Customer Order Handling** process model with the **Create self-transitions** option checked:



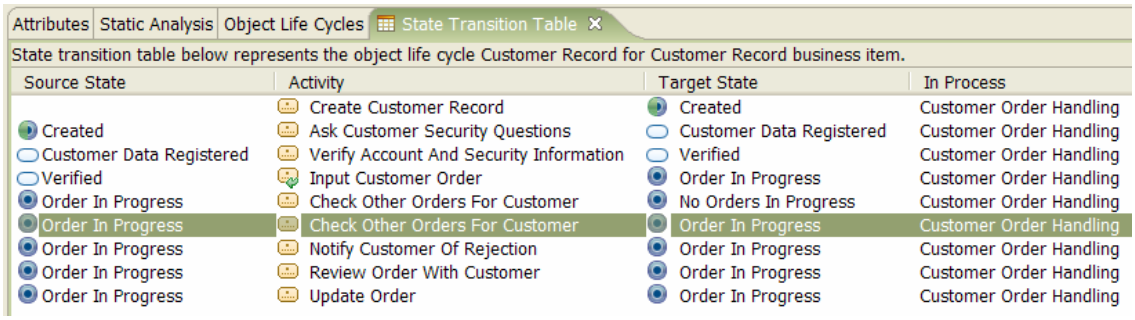
Open the extracted **Customer Record** life cycle. In this life cycle, the state **Order In Progress** has several self-transitions.

I

You may need to drag the transitions out by their bend-points to see all of them:



B

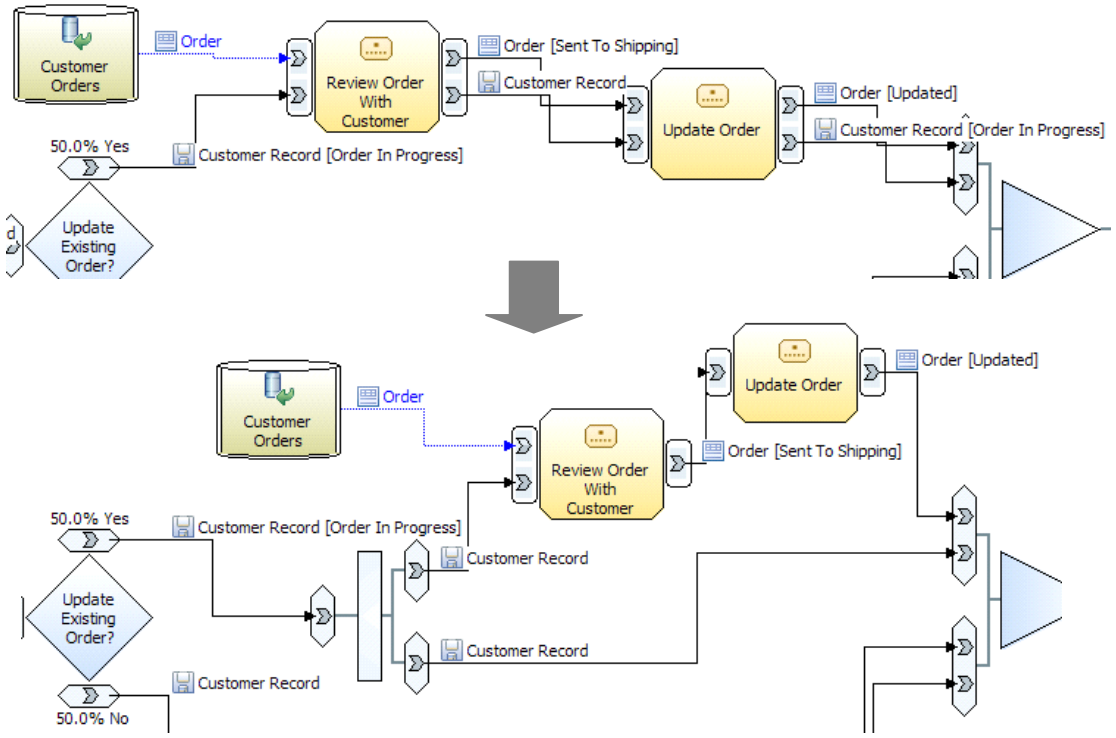


For each activity that corresponds to a self-transition, you should ensure that the activity requires the business item as an input and an output. Generally, only those activities that create or update a business item should have an output of that business item. In high-level process models, business items are often routed through all activities. However, since activity inputs and outputs influence service interfaces during the design and implementation phases, you should avoid unnecessary outputs and route business items through control nodes or repositories where possible.

Examine each activity associated with a self-transition as follows:

- Should this activity have the business item as an output?
    - If yes, should it leave the business item in the same state?
        - If yes, do nothing.
        - If no, add another state to the activity output.
    - If no, remove the object output and adjust object flows.

In this example there are four self-transitions. The **Review Order With Customer** task requires the **Customer Record** to retrieve the correct **Order**, but this task does not update the **Customer Record** and therefore should not have it as an output. The **Update Order** task does not actually require to access the **Customer Record** at all. The process model should be adapted as follows:

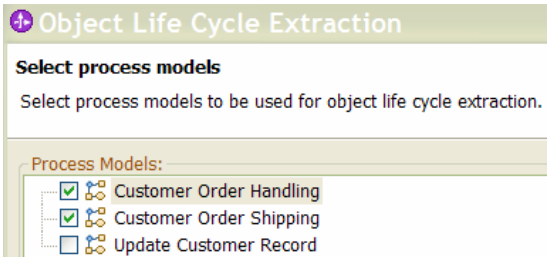


The **Notify Customer Of Rejection** task does make some updates to the **Customer Record** business item to add the negative credit check to the customer history. This task however does not change the state of the **Customer Record** and therefore the process model can be left as is. The **Check Other Orders For Customer** task in the **Update**

Customer Record global process either updates the state of a **Customer Record** to **No Orders In Progress**, or leaves it in the **Order In Progress** state. No changes are required to this task either.

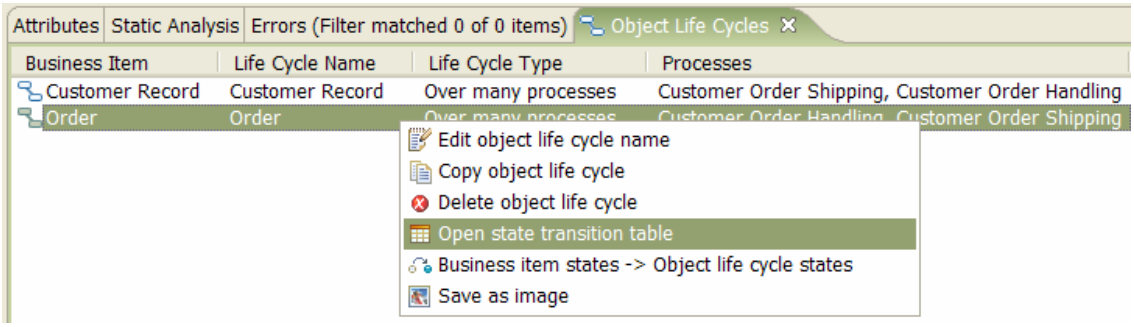
Generating object life cycles from several process models

In order to elicit end-to-end life cycles of business items, you need to ensure that the object life cycle extraction traverses all the processes that manipulate the business item at hand. Apart from the **Customer Order Handling** process model, the **Order** and **Customer Record** business items are also manipulated in the **Customer Order Shipping** process model. Run the object life cycle extraction again with both of these process models selected and ensure that the **Traverse local and global processes** option is used.

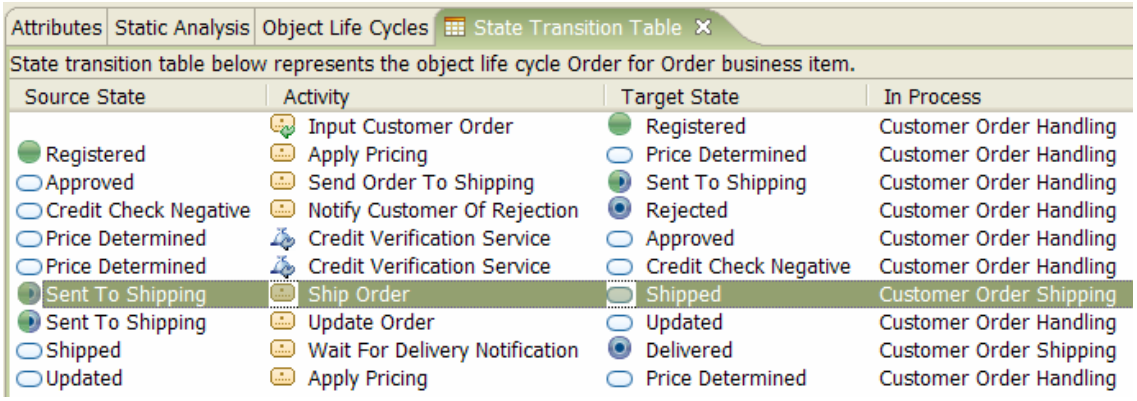


Open the generated object life cycle for the **Order** business item:

This object life cycle can also be examined using the **State Transition Table** view, especially suited for a compact visualization of complex life cycles. Right-click on the entry for the **Order** object life cycle in the Object Life Cycles view and select **Open state transition table** (in the Business edition, you can also double-click on the entry).



If you are using the business edition of Object Life Cycle Explorer, you should already be familiar with the State Transition Table view. Each row in the State Transition Table view represents a transition in the object life cycle, which is associated with some activity. The source and target states are shown in the first and third columns, respectively. Green circles are used to mark the targets of initial transitions and blue circles with a dark blue dot in the middle are used to mark the source states of final transitions. The icon next to the activity name in the second column shows what type of activity it is, i.e. local task, global task, global service, etc. The fourth column gives the name of the process model from which the transition was extracted. The rows can be sorted by the contents of each column by clicking on the corresponding column label.



Following the 5-step Object Life Cycle Validation method to examining the life cycle, you may identify the initial transition to state **Sent To Shipping** and the final transition from state **Sent To Shipping** as invalid. Since **Sent To Shipping** is a last state for the **Customer Order Handling** process model and it is a first state for the **Customer Order Shipping** process model, the object life cycle extraction introduces the transitions from the initial state and to the final state in the generated life cycle.

B

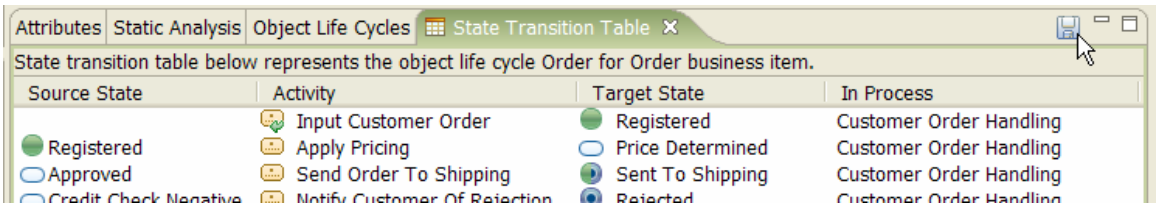
You can remove these transitions and make other changes to an extracted life cycle using the Object Life Cycle editor and save it under a different name. The changes you make to an object life cycle are not reflected in the original process models.

Exporting object life cycles

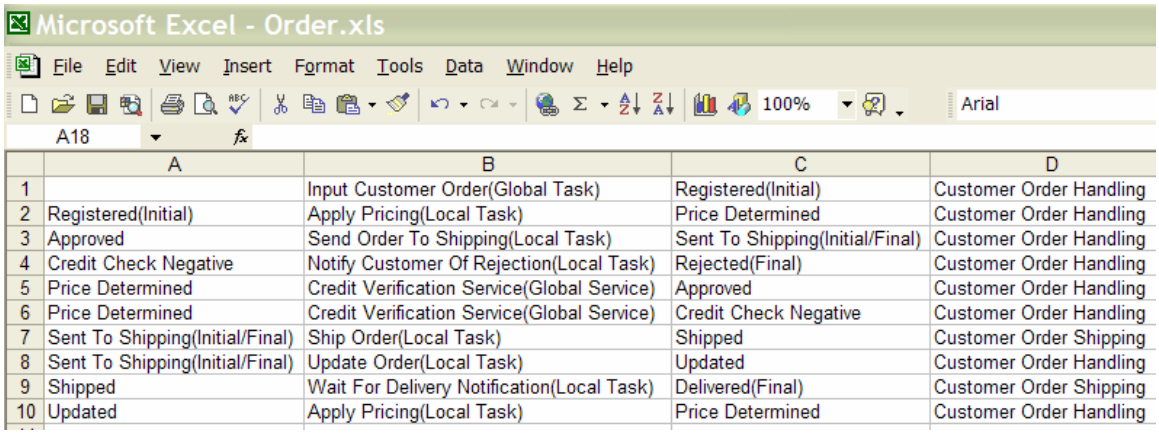
Object life cycles can be exported in several formats: table format, as an image, or as an archive for project interchange.

Table format export

The contents of a state transition table can be saved in a tab delimited text file, which can be opened in Excel as a table. In the State Transition Table view for the **Order** life cycle, click on the **Save** icon in the upper-right corner. In the file dialog box, navigate to the location where you want to save the file. The file is saved as .xls by default.



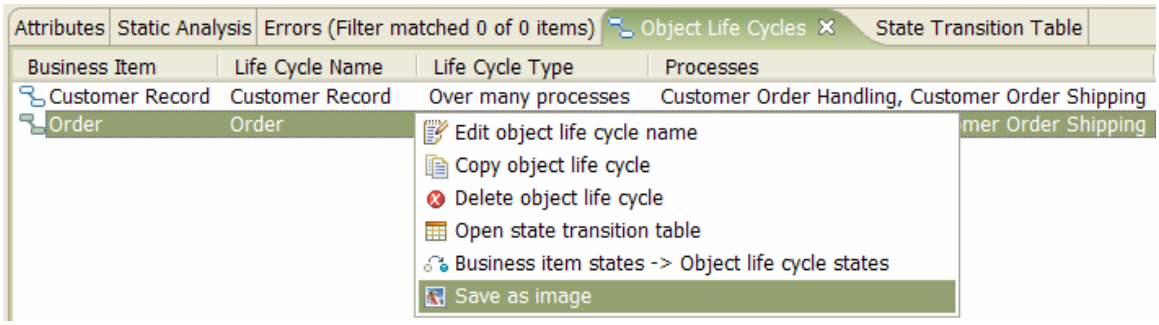
You can open the exported file in Excel and use it for reports.



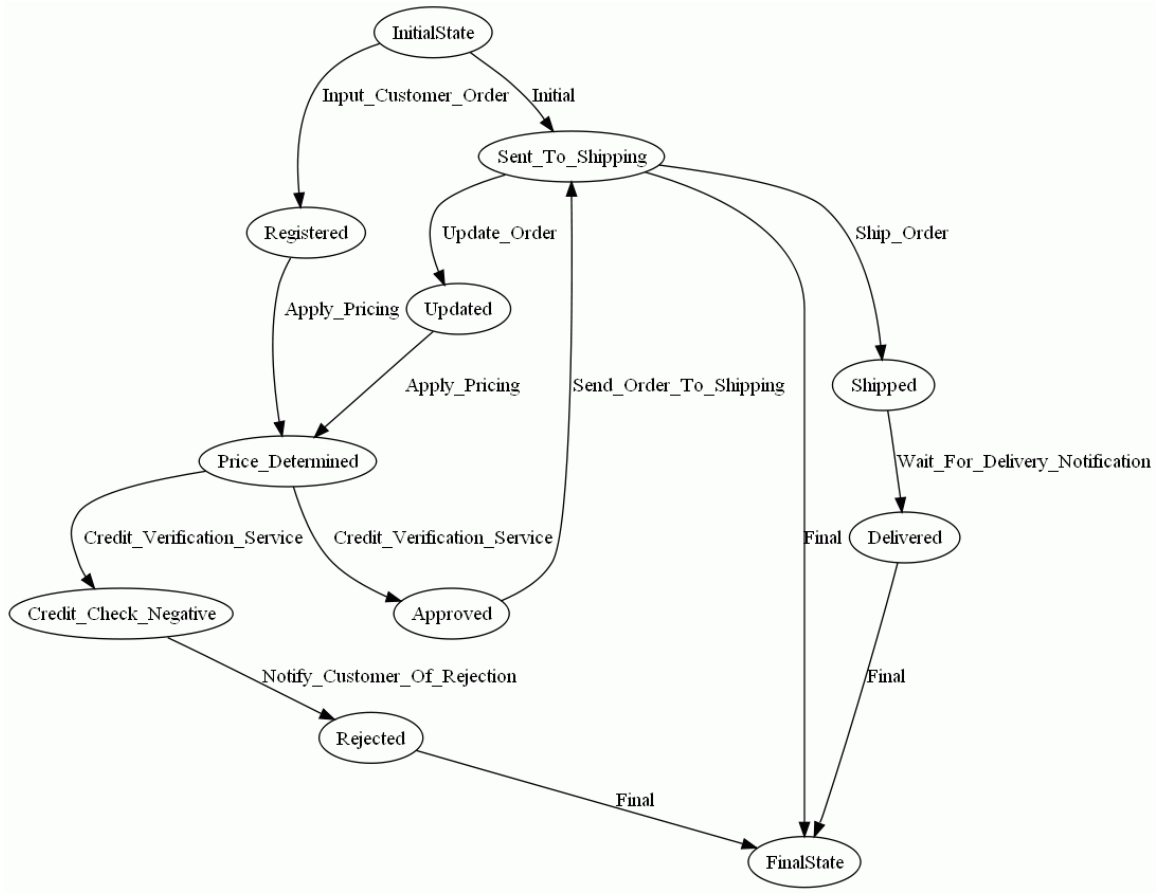
	A	B	C	D
1		Input Customer Order(Global Task)	Registered(Initial)	Customer Order Handling
2	Registered(Initial)	Apply Pricing(Local Task)	Price Determined	Customer Order Handling
3	Approved	Send Order To Shipping(Local Task)	Sent To Shipping(Initial/Final)	Customer Order Handling
4	Credit Check Negative	Notify Customer Of Rejection(Local Task)	Rejected(Final)	Customer Order Handling
5	Price Determined	Credit Verification Service(Global Service)	Approved	Customer Order Handling
6	Price Determined	Credit Verification Service(Global Service)	Credit Check Negative	Customer Order Handling
7	Sent To Shipping(Initial/Final)	Ship Order(Local Task)	Shipped	Customer Order Shipping
8	Sent To Shipping(Initial/Final)	Update Order(Local Task)	Updated	Customer Order Handling
9	Shipped	Wait For Delivery Notification(Local Task)	Delivered(Final)	Customer Order Shipping
10	Updated	Apply Pricing(Local Task)	Price Determined	Customer Order Handling

Image format export

Provided that you have **dot** from **Graphviz** installed (see installation instructions in Section 2), you can generate an image file for a selected object life cycle. Right-click on the entry for the **Order** object life cycle in the Object Life Cycles view and select **Save as image**.



In the file dialog box, navigate to the location where you want to save the image file. You can then open the generated image from the file system.

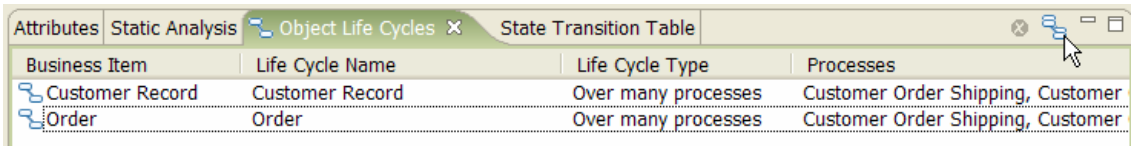


The image export feature using dot is provided predominantly for the Business edition of Object Life Cycle Explorer, which does not include the Object Life Cycle editor.

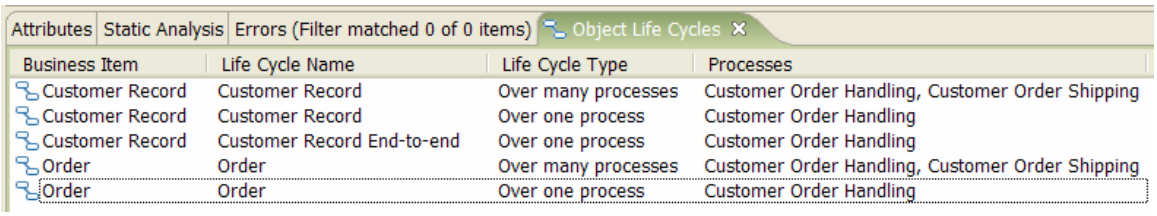
Archive export for project interchange

You can export all object life cycles in the current project as an archive, which can be subsequently imported into a different WebSphere Business Modeler project.

Examine what life cycles you have in your project first. In the Object Life Cycles view, click on the **Show all object life cycles for project** button .

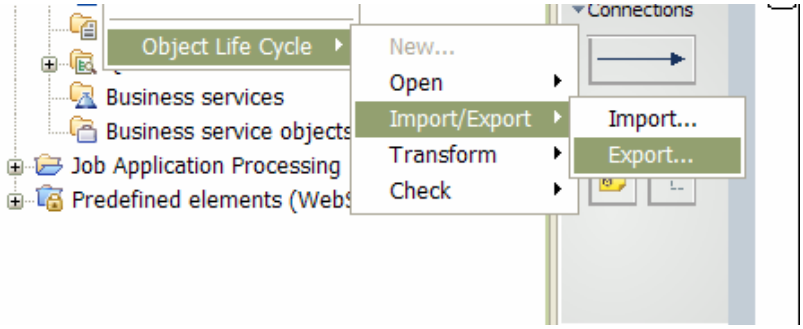


This opens up all the object life cycles in your current project. Running the archive export will export all of these object life cycles.



If you want to delete some object life cycles before exporting, select the corresponding entries in the Object Life Cycles view and click on the **Delete object life cycle** button .

Once you are ready to export, right-click on one of the elements contained in your project in the Project Tree. Select **Object Life Cycle → Import/Export → Export…**



Navigate to the location where you want the archive to be saved and click on **Save**. A .zip archive is created. You can import this archive in another project by selecting **Object Life Cycle → Import/Export → Import…**

Elicitation of end-to-end object life cycles demonstrated in this tutorial can be used for harvesting object life cycles as reusable modeling assets. An object life cycle harvested in one project can be used as a *reference* object life cycle in another project. Continue to **Tutorial 2** to learn more about reference-driven process modeling.

5. Tutorial 2: Reference-driven Process Modeling

This tutorial predominantly demonstrates the use of features available in the Integration edition of Object Life Cycle Explorer only.

This tutorial will show you how to:

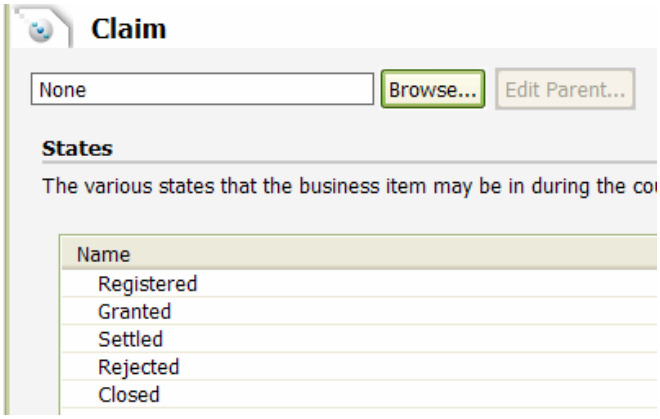
- import an object life cycle archive into a project,
- create a new object life cycle,
- generate a process model from one or more object life cycles,
- check consistency of a process model against given object life cycles.

In certain scenarios, you may want to obtain a process model that is based on some already existing object life cycles, referred to as *reference* object life cycles. A reference object life cycle may embody an industry standard or a best practice; it may be harvested from previous projects or developed to capture the requirements of the current project. In this tutorial, you will use reference object life cycles to create a process model for handling insurance claims.

Create a new Business Modeling Project called **Claim Handling**. At this stage you are not aware of all the detailed steps that a claim handling process comprises, but you have already elicited the requirements for the state evolution of a single claim by speaking to the domain experts. You can capture these requirements in an object life cycle, which will serve as a reference for further process modeling.

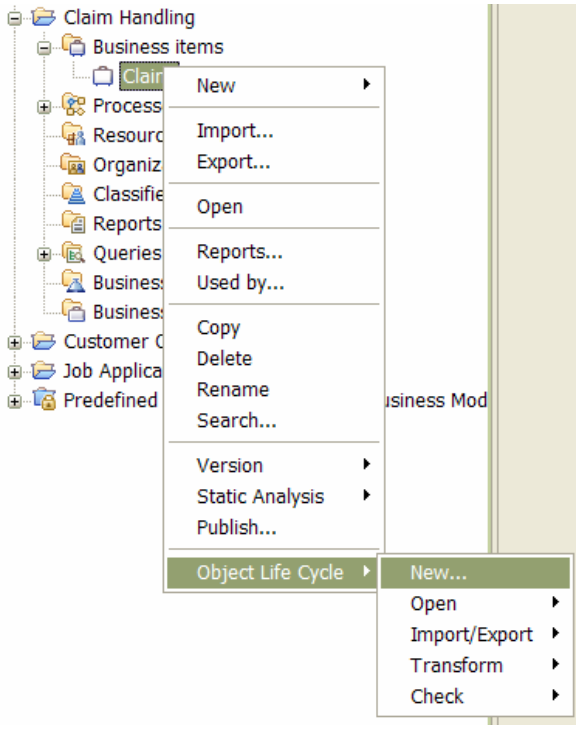
Creating a new object life cycle

Create a new business item called **Claim** and add states to this business item to obtain the following result:

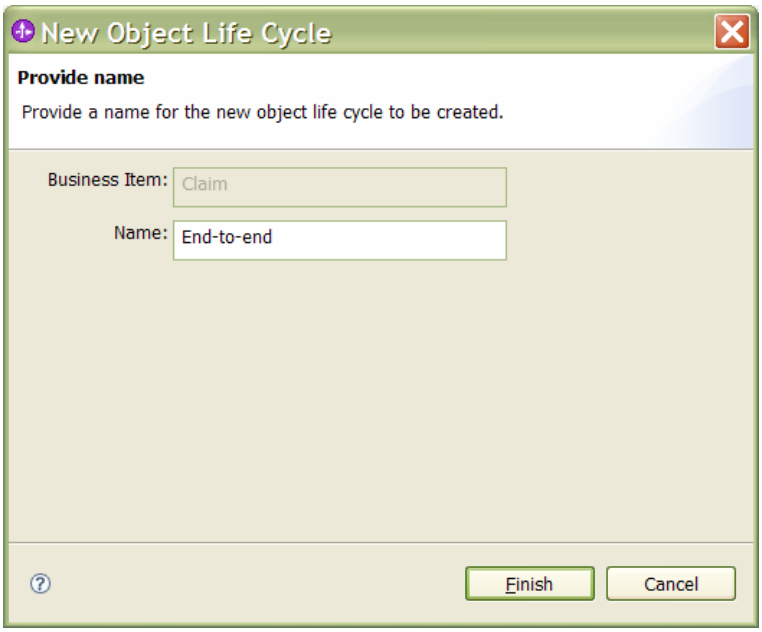


Save the **Claim** business item.

Right-click on the **Claim** business item in the Project Tree and select **Object Life Cycle → New…**

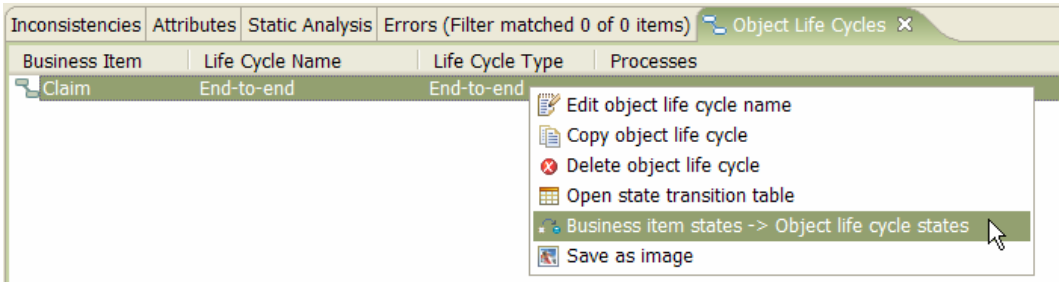


In the **New Object Life Cycle** wizard, enter **End-to-end** as the name for the new object life cycle and click on **Finish**.

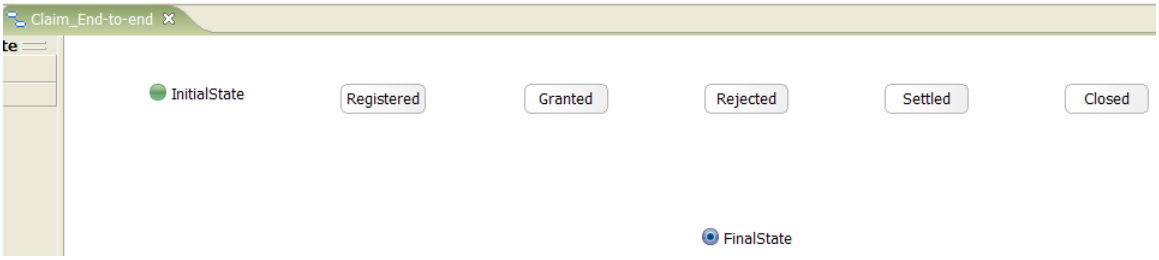


The Object Life Cycle editor is opened with an initial and a final state being the only elements in the new life cycle. The newly created object life cycle is also shown in the Object Life Cycles view.

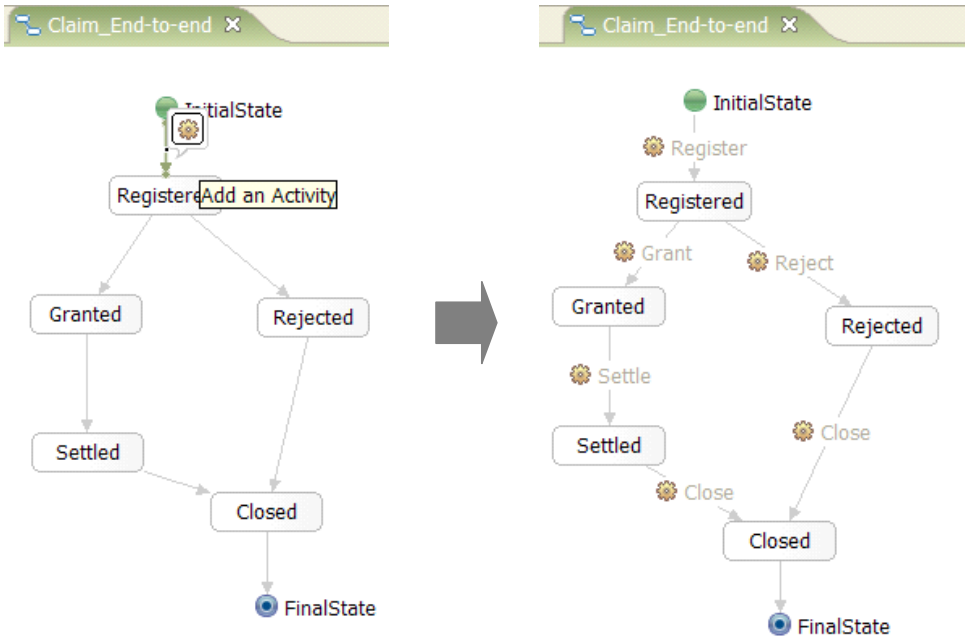
To simplify the modeling of an object life cycle, you can automatically add the states defined for this business item to the Object Life Cycle editor. Right-click on the **Claim** life cycle entry in the Object Life Cycles view and select **Business item states -> Object life cycle states**.



The states that you defined for the **Claim** business item now appear in the Object Life Cycle editor.



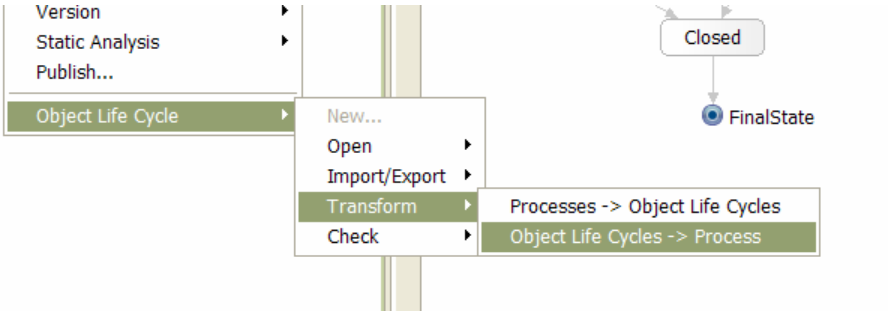
Connect the states with transitions and then associate an activity with each transition:



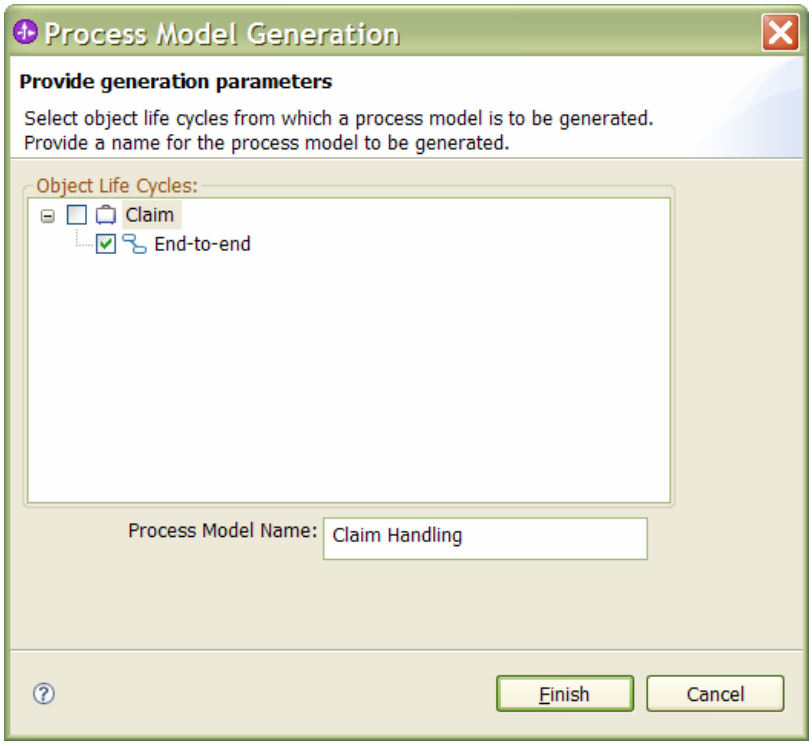
Save the modeled object life cycle.

Generating a process model from one object life cycle

You can now automatically generate a process model from the **Claim** life cycle. Right-click on some element of the **Claim Handling** project in the Project Tree and select **Object Life Cycle → Transform → Object Life Cycles -> Process**.

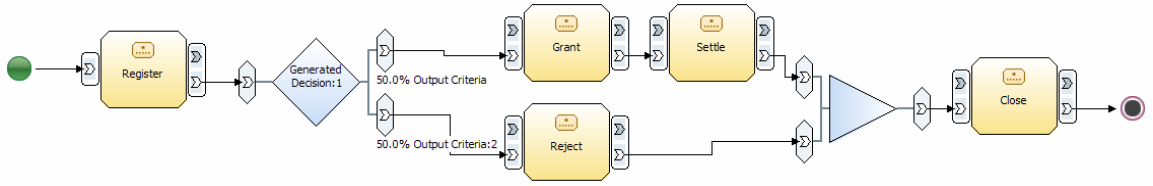


In the **Process Model Generation** wizard, select the **End-to-end** life cycle of the **Claim** business item and provide **Claim Handling** as the name for the process model to be generated. Click on **Finish**.

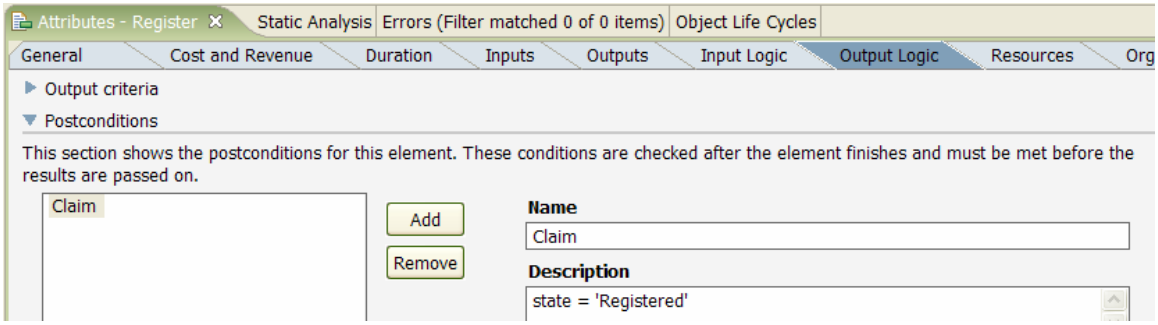


A new process called **Claim Handling** is added to the Project Tree.

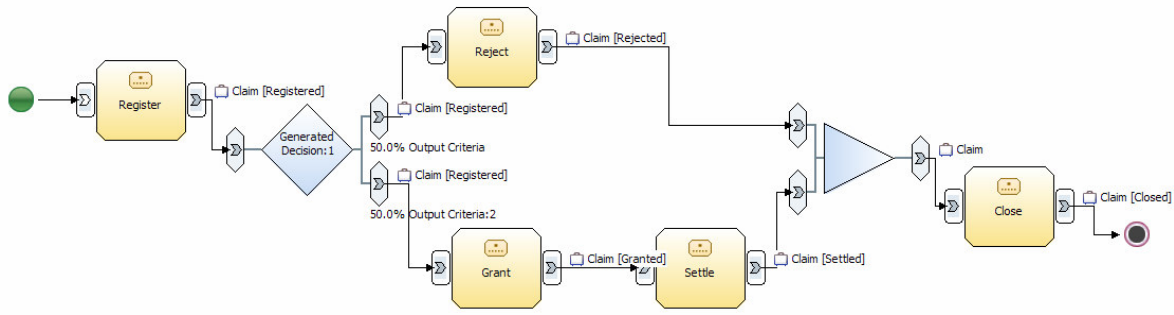
Open the newly generated **Claim Handling** process model:



You can see that a task is generated for each activity associated with a transition in the object life cycle. The tasks in the process model are connected with control flows. The tasks have inputs and output of business item **Claim**, but these are not connected. Although these inputs and outputs are not associated with state information, the input and output states of business items are stored in the pre/postconditions of the tasks. For example, go to the **Attributes** of the **Register** task and open the **Output Logic** tab. Click on **Claim** in the **Postconditions**. The **Description** field of this postcondition indicates that the state of **Claim** is **Registered** after this task executes. See the section on **Advanced Topics** for more details about specifying business item states using pre/postconditions.

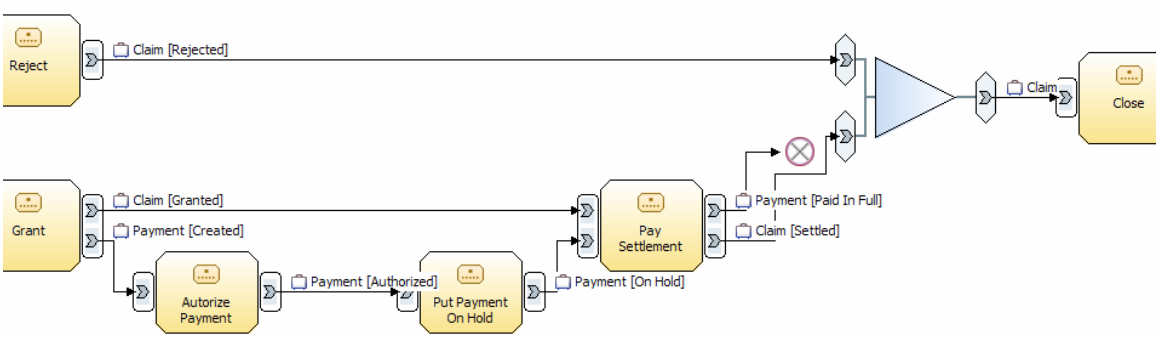


A process model generated from a life cycle only serves as a skeleton and needs to be further customized to your needs. For example, object flows and state specifications need to be added. Edit the generated process model as shown below and save it as **Claim Handling With Object Flow**. Alternatively, you can import this process model by importing the **Claim Handling With Object Flow.mar** file from the **samples** directory into the **Claim Handling** project (do not overwrite the **Claim** business item during the import).



The **Claim Handling With Object Flow** process model currently only deals with the **Claim** business item. However, since a payment needs to be made to the claimant when the claim is granted, this process model should also manipulate a **Payment** business item.

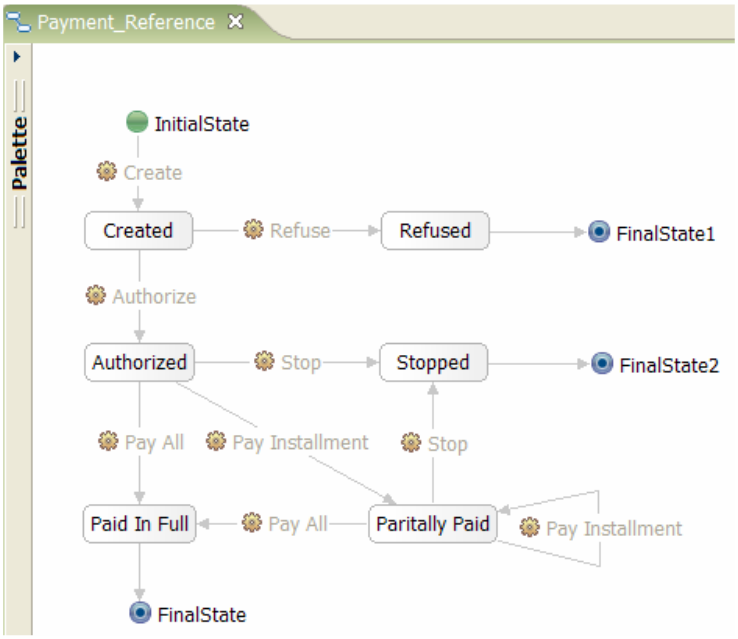
Copy the **Claim Handling With Object Flow** process model and rename it to **Claim and Payment Handling**. Create a new **Payment** business item, add the states **Created**, **Authorized**, **On Hold** and **Paid In Full** to it, and then adapt the part of the process model between the tasks **Grant** and **Close** as shown below. Alternatively, you can import this process model by importing the **Claim And Payment Handling.mar** file from the **samples** directory into the **Claim Handling** project (do not overwrite the **Claim** business item during the import).



You will now check whether this process model is consistent with a reference life cycle for **Payment**, which has been defined as the industry best practice.

Importing object life cycles

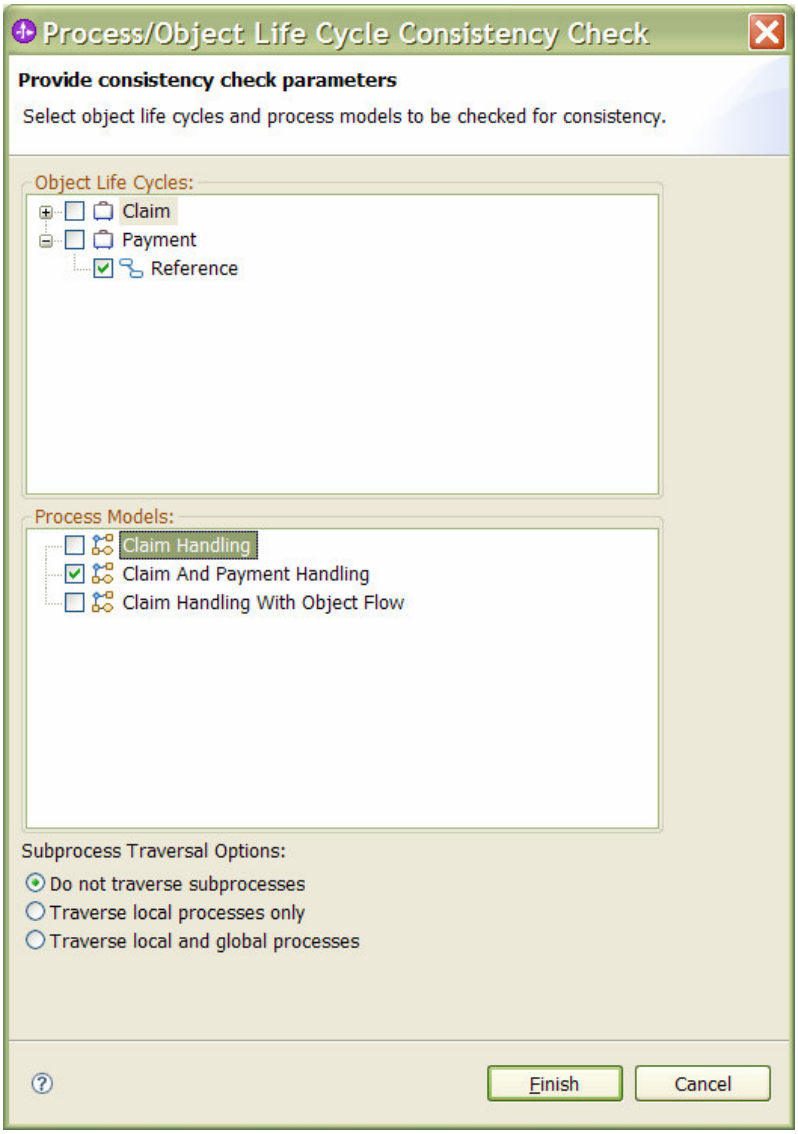
In the Project Tree, right-click on some element in the **Claim Handling** project and select **Object Life Cycle → Import/Export → Import…** In the file dialog box, navigate to the **Payment Reference OLC.zip** archive in the **samples** directory and click on **Open**. An entry for the imported **Reference** life cycle for business item **Payment** is shown in the Object Life Cycles view. Open this life cycle in the Object Life Cycle editor:



Checking consistency of a process model against object life cycles

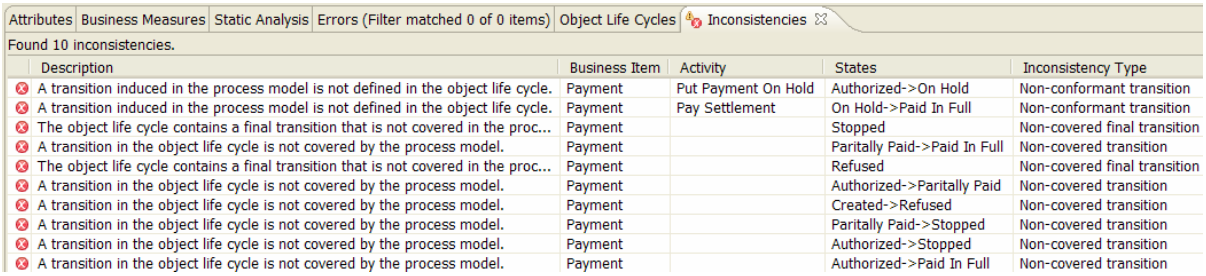
Check whether the **Claim And Payment Handling** process model that you created is consistent with this reference life cycle. Right-click on some element of the **Claim Handling** project in the Project Tree and select **Object Life Cycle → Check → Process/Object Life Cycle Consistency**.

In the **Process/Object Life Cycle Consistency Check** wizard, select the **Reference** object life cycle for the **Payment** business item and the **Claim And Payment Handling** process model and click on **Finish**.



An **Inconsistencies** view opens with a list of detected inconsistencies. Inconsistencies fall into two categories: *non-conformance* and *non-coverage*. A non-conformance inconsistency indicates that some manipulation of the business item does not conform to the given object life cycle. A non-coverage inconsistency is detected when some parts of the given object life cycle are not covered by the process model.

In this example, two **Non-conformant transition** inconsistencies are discovered. As stated in the **Description** field of the first such inconsistency (the order in which inconsistencies are displayed may vary), there is a transition from state **Authorized** to state **On Hold** induced in the process model that is not defined in the object life cycle. Such an inconsistency can be resolved in several ways. For example, by removing the associated activity, **Pay Settlement** in this case, from the process model. Alternatively the input and/or output states of the activity can be adjusted, so that it does not induce this particular transition anymore. Object flow can also be routed differently to ensure that this transition is not induced.


Attributes | Business Measures | Static Analysis | Errors (Filter matched 0 of 0 items) | Object Life Cycles | Inconsistencies

Found 10 inconsistencies.

Description	Business Item	Activity	States	Inconsistency Type
A transition induced in the process model is not defined in the object life cycle.	Payment	Put Payment On Hold	Authorized->On Hold	Non-conformant transition
A transition induced in the process model is not defined in the object life cycle.	Payment	Pay Settlement	On Hold->Paid In Full	Non-conformant transition
The object life cycle contains a final transition that is not covered in the proc...	Payment		Stopped	Non-covered final transition
A transition in the object life cycle is not covered by the process model.	Payment		Paritally Paid->Paid In Full	Non-covered transition
The object life cycle contains a final transition that is not covered in the proc...	Payment		Refused	Non-covered final transition
A transition in the object life cycle is not covered by the process model.	Payment		Authorized->Paritally Paid	Non-covered transition
A transition in the object life cycle is not covered by the process model.	Payment		Created->Refused	Non-covered transition
A transition in the object life cycle is not covered by the process model.	Payment		Paritally Paid->Stopped	Non-covered transition
A transition in the object life cycle is not covered by the process model.	Payment		Authorized->Stopped	Non-covered transition
A transition in the object life cycle is not covered by the process model.	Payment		Authorized->Paid In Full	Non-covered transition

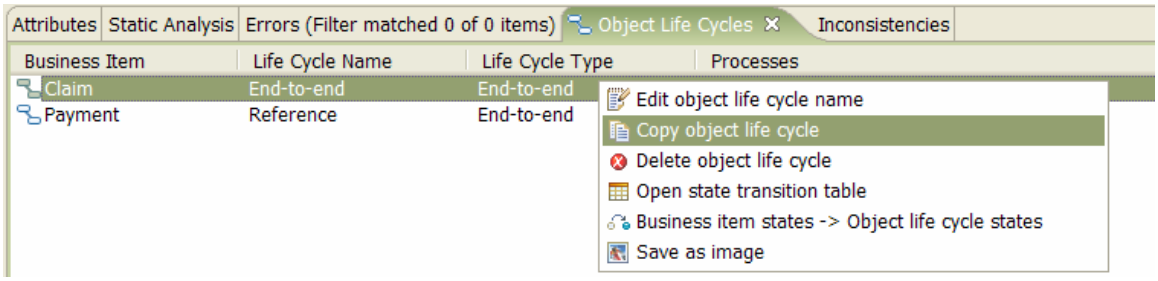
In this example, the majority of the discovered inconsistencies are **Non-covered transitions**. Transitions contained in the **Reference** object life cycle, e.g. from **Authorized** to **Partially Paid** or from **Authorized** to **Stopped**, are not induced and therefore not covered in the **Claim And Payment Handling** process model. Adding activities to the process model can resolve such inconsistencies. Alternatively, inputs and/or outputs of the existing activities can be adjusted, so that they induce the non-covered transitions.

A complete alignment with the **Reference** life cycle for **Payment** would require you to manually resolve all these inconsistencies. When you are at the beginning stages of designing a process model, you can achieve consistency with several object life cycles by automatically generating a process model from them.

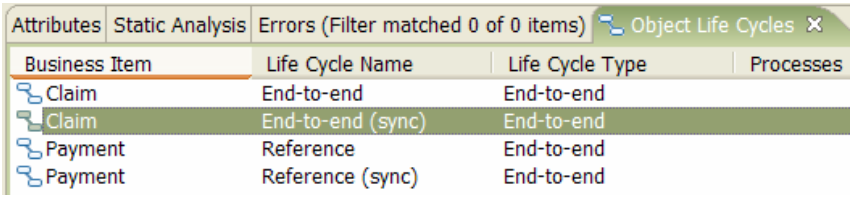
Generating a process model from several object life cycles

Whenever the process model generation is performed with more than one object life cycle as input, *synchronizing activities* in the object life cycles first need to be identified. Associating a synchronizing activity with transitions in different life cycles indicates that these transitions must occur at the same time. During process model generation, one activity that changes the state of several business items is generated from such transitions.

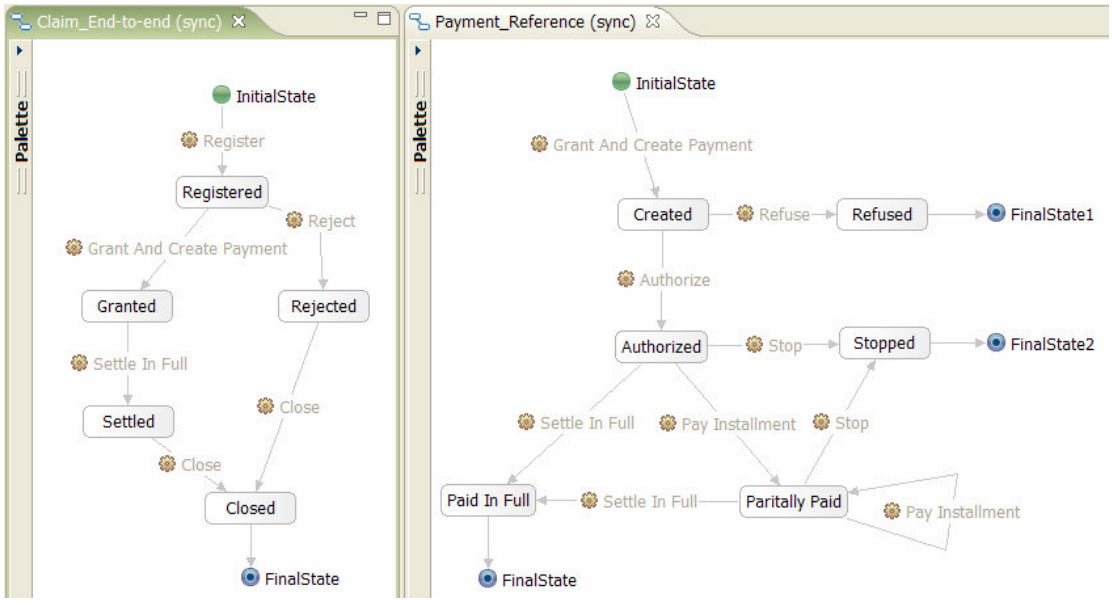
Create copies of the **End-to-end** life cycle for **Claim** and the **Reference** life cycle for **Payment**. Right-click on an object life cycle entry in the Object Life Cycles view and select **Copy object life cycle**.



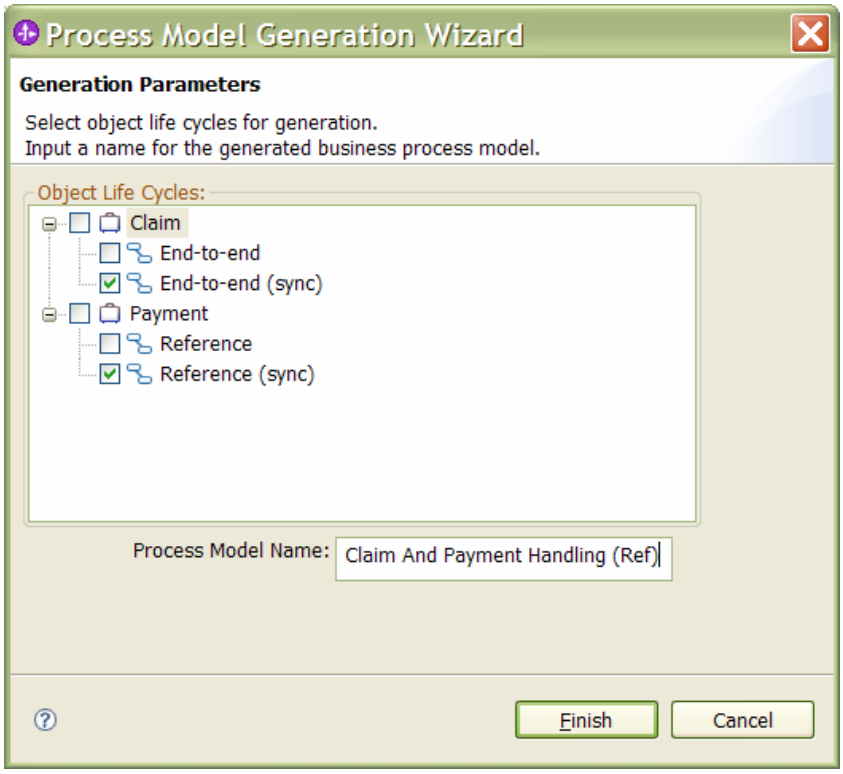
Name the newly created copies **End-to-end (sync)** and **Reference (sync)**. Note that the formatting of the object life cycles in the Object Life Cycle editor is currently lost during the copying process.



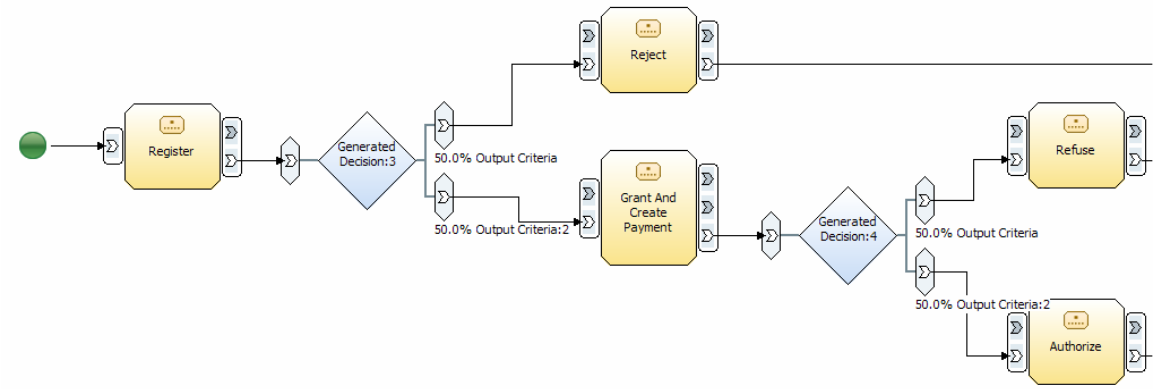
Since the **Payment** should only be **Created** if the **Claim** is **Granted**, the **Grant** and **Create** activities in the two life cycles need to be synchronized. This is done by renaming the activities to the same name in both object life cycles. Rename activity **Grant** in the **Claim** life cycle and activity **Create** in the **Payment** life cycle to **Grant And Create Payment**. Additionally, a **Claim** can be **Settled** only once the full **Payment** has been made and thus you need to synchronize the **Settle** and **Pay All** activities. Indicate this by renaming these two activities to **Settle In Full**.



Invoke the process model generation. In the **Process Model Generation** wizard, select the **End-to-end (sync)** life cycle for **Claim** and the **Reference (sync)** life cycle for **Payment**. Provide **Claim And Payment Handling (Ref)** as the process name.



The generated process model now covers both object life cycles:



You have now obtained a process model for the handling of claims, which is based on two reference object life cycles. This generated process model can now be further customized, as necessary.

6. Resolving Inconsistencies between Process Models and Object Life Cycles

new in V1.1.0

This section explains the use of the inconsistency resolution support available in the Integration edition of Object Life Cycle Explorer only. The section assumes that you have already completed Tutorials 1 and 2 and are familiar with most of the features.

This section will explain to you how to:

- semi-automatically resolve inconsistencies between process models and object life cycles,
- choose among alternative resolution options using side-effect information.

In Tutorial 2, you used the consistency checking feature to automatically detect inconsistencies between given process models and object life cycles. Manual resolution of inconsistencies can be very time-consuming. Most importantly, you need to identify how to change the models to resolve a particular inconsistency, but at the same time you also need to ensure that these changes do not introduce new inconsistencies as a side-effect. Object Life Cycle Explorer (version 1.1.0 and higher) simplifies the process of inconsistency resolution by offering you a range of options for resolving a particular inconsistency, called *resolutions*. By selecting one of the offered resolutions, changes are automatically applied to the models to resolve the inconsistency. The most beneficial resolutions that resolve many inconsistencies and introduce few new ones are identified to assist you in choosing among the alternatives.

In the following, we first explain in some detail the types of inconsistencies that can arise between process models and object life cycles. We then introduce the set of supported resolutions that can be applied to resolve inconsistencies of these types.

Inconsistency types and resolutions

As already explained in Tutorial 2, inconsistencies are grouped into two main categories, namely non-conformance and non-coverage. Each category contains 3 inconsistency types, as described below.

Non-conformance inconsistency types:

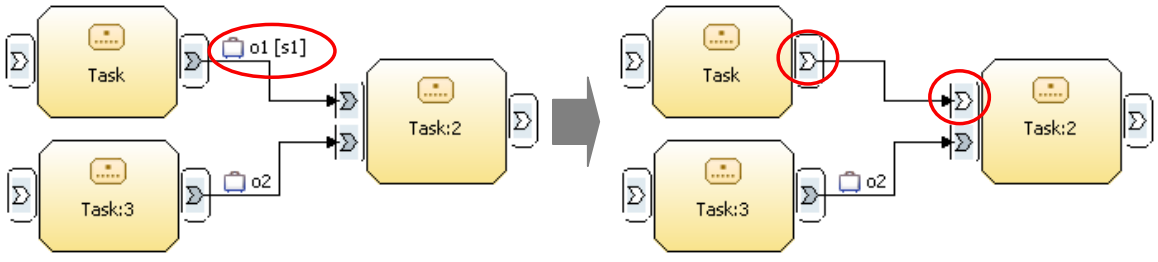
- *non-conformant transition:* a task/subprocess *t* in the given process model induces a transition from state *src* to state *tgt* on a business item *o*, but this transition is not defined in the object life cycle for *o*;
- *non-conformant initial transition:* a business item *o* is created in state *first* or received in state *first* via an input parameter, but there is no transition from the initial state to state *first* in the object life cycle for *o*;

- *non-conformant final transition:* state *last* is reached as the last state of a business item *o* during the execution of the given process model, but there is no transition from state *last* to a final state in the object life cycle for *o*.

Non-coverage inconsistency types:

- *non-covered transition:* a transition from state *src* to state *tgt* is defined in the object life cycle for business item *o*, but it is not induced by any task/subprocess in the given process model;
- *non-covered initial transition:* a state *init* has an incoming transition from the initial state in the object life cycle for business item *o*, but *o* is neither created nor received by the process model in state *init*;
- *non-covered final transition:* a state *fin* has an outgoing transition to a final state in the object life cycle for business item *o*, but *fin* is not a last state of *o* in the given business process model.

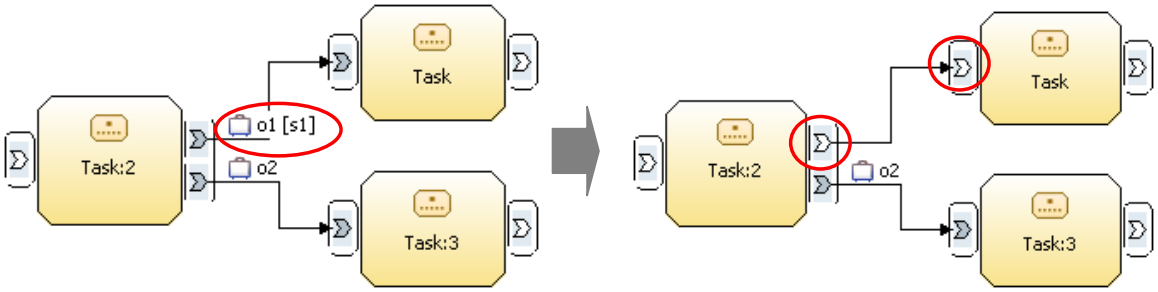
There are 8 resolutions currently supported in Object Life Cycle Explorer (version 1.1.0). Each resolution defines a change or a *transformation* of the process model associated with a particular inconsistency. For example, a non-conformant transition inconsistency caused by a task *t* that induces a transition for a business item *o* can be resolved by removing data from the incoming connections of *t* that are associated with *o*.
This resolution is labeled **R1** in the table below, which gives an overview of all the 8 resolutions that are currently supported. Each resolution is associated with a *context* comprising a process node, a business item, and sometimes also a state. This context defines where in the process model the transformation will be applied if you choose this resolution.

<table>
<tr><th></th><th>Resolution description and example:</th></tr>
<tr><td>R1</td><td>
<table>
<tr><td>Description:</td><td>Process Node:</td><td>Business Item:</td><td>State:</td></tr>
<tr><td>Remove data from incoming connections of this process node.</td><td>Task:2</td><td>o1</td><td>-</td></tr>
</table>


Data is removed from all incoming connections of Task:2 that are associated with business item o1.
</td></tr>
</table>

R2

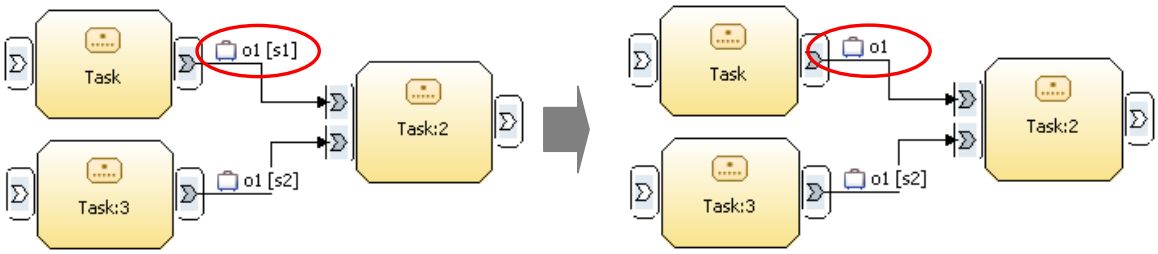
Description:	Process Node:	Business Item:	State:
Remove data from outgoing connections of this process node.	Task:2	o1	-



Data is removed from all outgoing connections of **Task:2** that are associated with business item **o1**.

R3

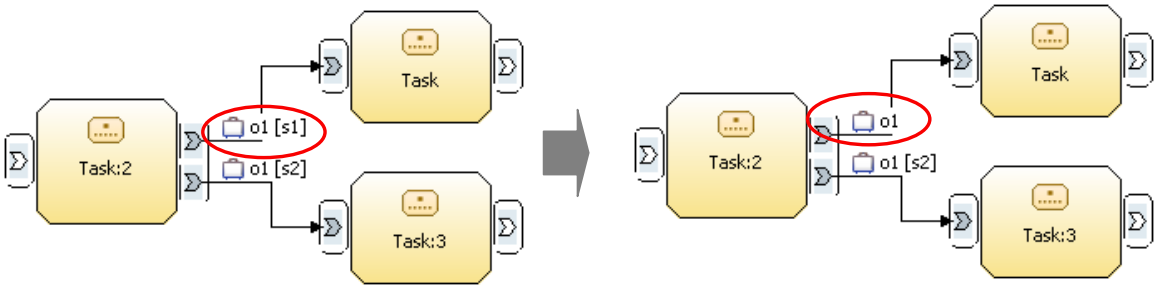
Description:	Process Node:	Business Item:	State:
Remove state from incoming connections.	Task:2	o1	s1



State **s1** is removed from an incoming connection of **Task:2** associated with business item **o1**.

R4

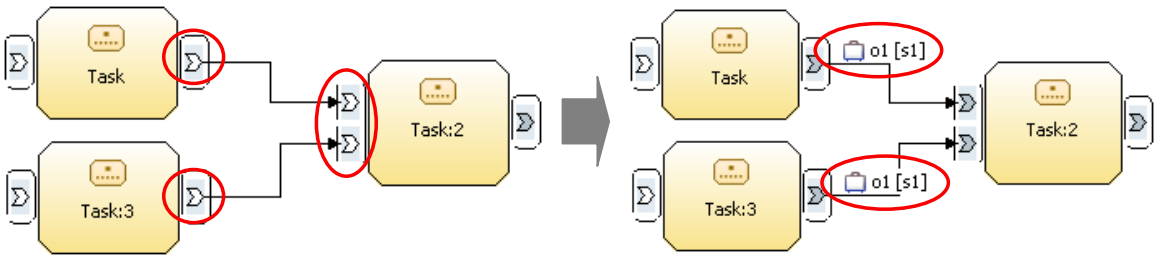
Description:	Process Node:	Business Item:	State:
Remove state from outgoing connections.	Task:2	o1	s1



State **s1** is removed from an outgoing connection of **Task:2** associated with business item **o1**.

R5

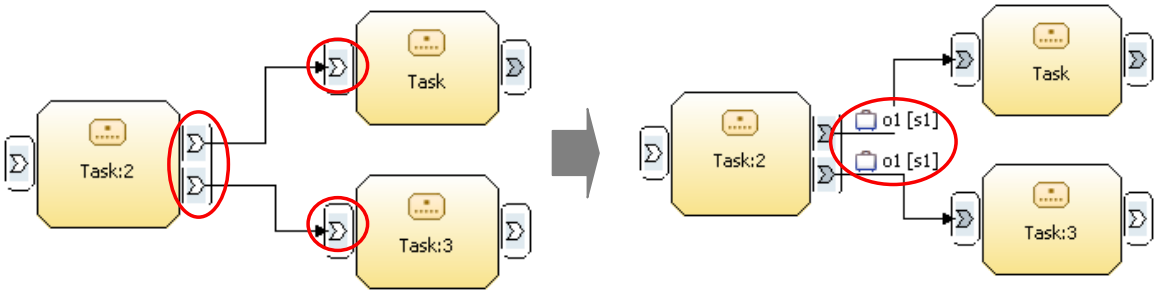
Description:	Process Node:	Business Item:	State:
Associate data and state with incoming connections of this process node.	Task:2	o1	s1



Business item **o1** and state **s1** are associated with incoming connections of **Task:2**.

R6

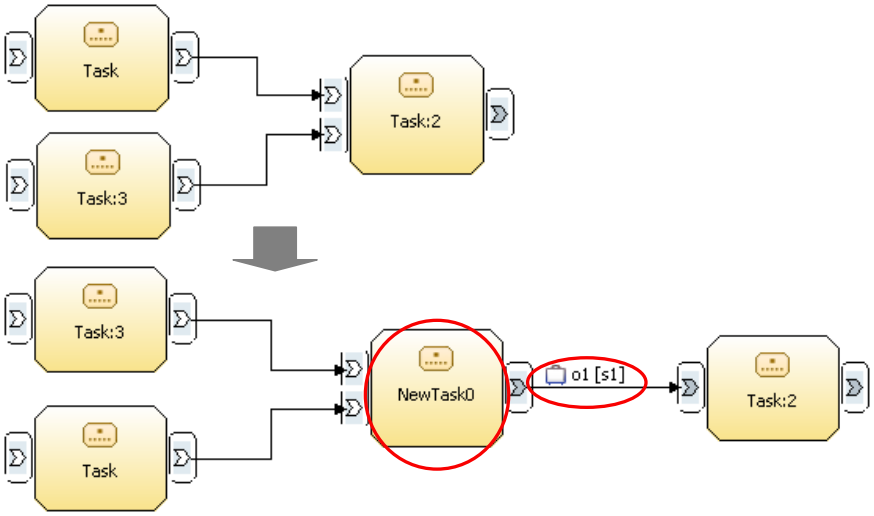
Description:	Process Node:	Business Item:	State:
Associate data and state with outgoing connections of this process node.	Task:2	o1	s1



Business item **o1** and state **s1** are associated with outgoing connections of **Task:2**.

R7

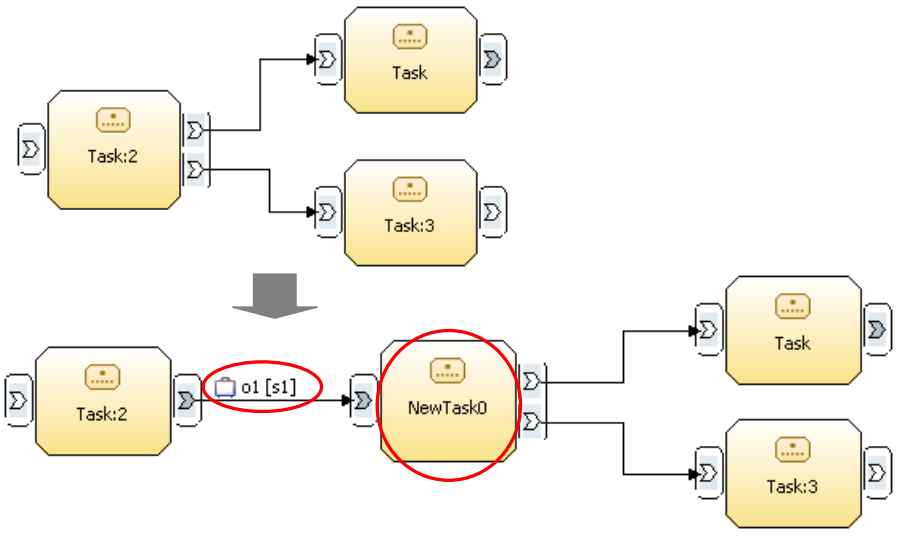
Description:	Process Node:	Business Item:	State:
Insert a new task between this process node and its predecessor nodes.	Task:2	o1	s1



A new task **NewTask0** is inserted between **Task:2** and its predecessor nodes, **Task:3** and **Task**. The connection introduced between **NewTask0** and **Task:2** is associated with business item **o1** and state **s1**.

R8

Description:	Process Node:	Business Item:	State:
Insert a new task between this process node and its successor nodes.	Task:2	o1	s1



A new task **NewTask0** is inserted between **Task:2** and its successor nodes, **Task:3** and **Task**. The connection introduced between **NewTask0** and **Task:2** is associated with business item **o1** and state **s1**.

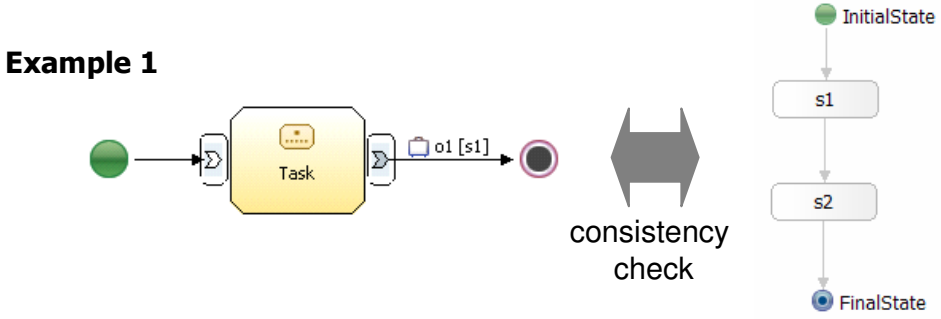
Some resolutions can be applied to resolve inconsistencies of different types. For example, **R1** can also be used to resolve a non-conformant final transition to state *s1* induced by *Task:2* for business item *o1*.

The table below shows the resolutions that can be applied to resolve inconsistencies of different types.

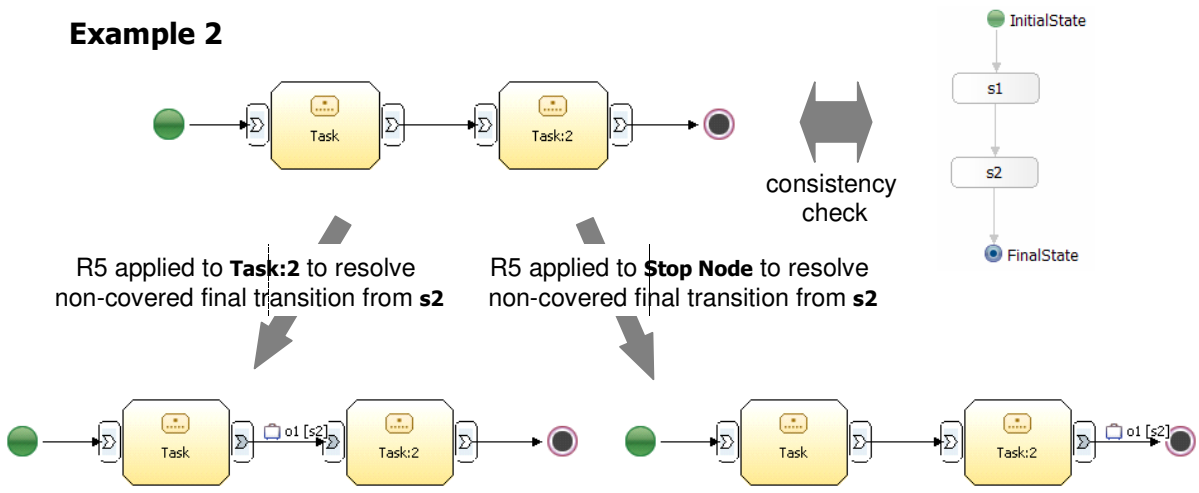
Inconsistency type / Resolution	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
non-conformant transition	+	+	+	+			+	+
non-conformant initial transition		+		+	+		+	+
non-conformant final transition	+		+			+	+	+
non-covered transition					+	+	+	+
non-covered initial transition	+					+	+	+
non-covered final transition		+			+		+	+

Given an inconsistency of a particular type, it does not however always mean that all the resolutions associated with this inconsistency type (as shown in the above table) are *applicable*. The given process model determines which resolutions are applicable in each case. Let us consider a simple example to see this.

In **Example 1**, there are several inconsistencies between the process model and the object life cycle for business item **o1**, including a non-conformant final transition from state **s1** induced by the **Stop Node**. This inconsistency can be resolved by **R1** (removing data from the incoming connection of the **Stop Node**) or **R3** (removing the state **s1** from the incoming connection of the **Stop Node**). However, the outgoing connections of the **Stop Node** cannot be associated with data and state, since the **Stop Node** has no outgoing connections. Therefore, **R6** is not applicable.



In some cases, one resolution can be applied to different contexts within a process model to resolve the same inconsistency. In **Example 2**, one of the detected inconsistencies is a non-covered final transition from state **s2** for business item **o1**. **R5** can be applied in two contexts to resolve this inconsistency: either to associate **o1** and **s2** to the incoming connection of **Task:2** or to the incoming connection of the **Stop Node**.



As demonstrated in the above examples, one inconsistency can often be resolved by applying different resolutions or by applying the same resolution to different contexts. In the following, we explain how Object Life Cycle Explorer ranks such alternatives to assist you in choosing the best resolution in each case.

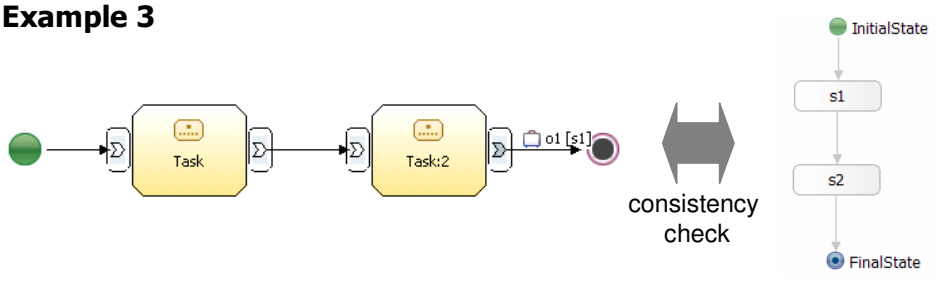
Resolution side-effects

Applying a resolution to resolve a particular inconsistency can often have *side-effects*, meaning that other inconsistencies can also be resolved as a by-product or new inconsistencies can be introduced. It is important to be aware of such side-effects before applying a resolution. In general, the resolution that removes the most inconsistencies should be chosen. Certainly, the changes made to the process model resulting by applying a resolution also need to be in agreement with the business requirements.

Running the consistency check on the process model and object life cycle in **Example 3** produces the 3 inconsistencies shown in the Inconsistencies view below. Along with the Inconsistencies view, the **Resolutions view** also opens on the completion of the consistency check[1]. When you select an inconsistency in the Inconsistencies view, the Resolutions view is populated with the applicable resolutions. **Example 3** shows that the non-covered final transition to state **s2** for business item **o1** can be resolved in two ways: by inserting a new task between the **Stop Node** and its predecessor nodes or between **Task** and its predecessor nodes.

[1] In the current version (V1.1.0), the Resolution view will only be opened if the consistency check is performed for one process model and one object life cycle.

Example 3



Inconsistencies view:

Attributes | Object Life Cycles | Inconsistencies

Found 3 inconsistencies.

Description	Process Node	Business Item	States	Inconsistency Type
An object reaches an illegal last state in the process with respect to the object life cy...	Stop Node	o1	s1	Non-conformant final transition
The object life cycle contains a final transition that is not covered in the process model.		o1	s2	Non-covered final transition
A transition in the object life cycle is not covered by the process model.		o1	s1->s2	Non-covered transition

Resolutions view:

Resolutions

Effect	Description	Process Node	Business Item	State
-3	Insert a new task between this process node and its predecessor nodes.	Stop Node	o1	s2
0	Insert a new task between this process node and its predecessor nodes.	Task	o1	s2

Resolutions view (continued):

Effect Description
Resolves(3): Non-covered transition(s1,s2) Non-covered final transition(s2) Non-conformant final transition(Stop Node,s1) ; Introduces(0): none
Resolves(1): Non-covered final transition(s2) ; Introduces(1): Non-conformant initial transition(NewTask0,s2)

The **Effect** column shows the overall effect of applying the resolution on the number of inconsistencies between the models. Therefore, the first resolution in the example above removes 3 inconsistencies (effect is -3) and the second resolution has no effect on the total number of inconsistencies (effect is 0). The resolutions are sorted according to their Effect values, and only those resolutions that decrease the number of inconsistencies are marked with a profit icon .

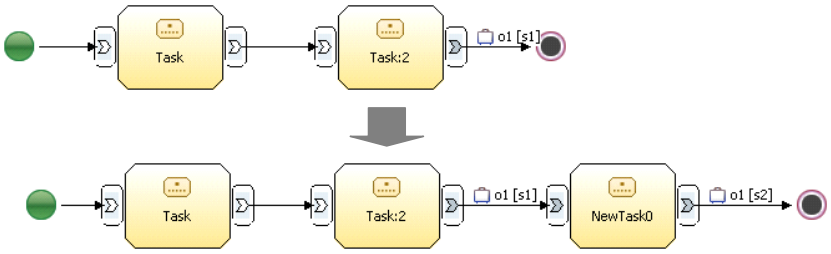
The details of the resolution side-effects are given in **Effect Description** column (see Resolutions view (continued) above). Here it can be seen how the overall effect is computed. For example, the second resolution resolves 1 inconsistency and introduces 1 new one, which is why its overall effect is 0.

A resolution can be applied by right-clicking it in the Resolutions view and selecting **Apply** from the context menu. The process model and the Inconsistencies view are automatically updated as a result.

The applications of the two resolutions to in **Example 3** are shown in the following diagram. The first resolution allows us to remove all 3 inconsistencies in one click, while the second resolution only replaces the non-covered final transition inconsistency with a non-conformant initial transition inconsistency. Naturally, the first resolution is the better choice here.

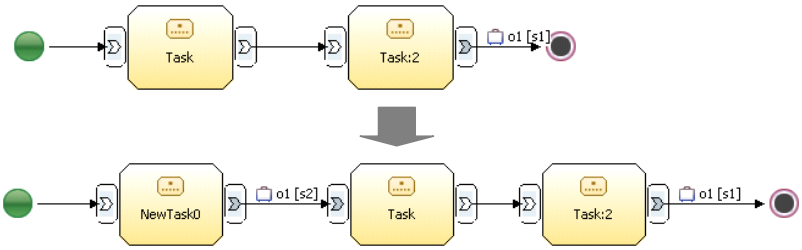
Example 3 (continued)

(1) Insert a new task between **Stop Node** and its predecessor nodes:



All inconsistencies resolved

(2) Insert a new task between **Stop Node** and its predecessor nodes:



Updated inconsistency view:

Attributes - Input | Object Life Cycles | Inconsistencies | Servers

Found 3 inconsistencies.

Description	Process Node	Business Item	States	Inconsistency Type
An object is created or received in an illegal state with respect to the object life cycle.	NewTask0	o1	s2	Non-conformant initial transition
An object reaches an illegal last state in the process with respect to the object life cycle.	Stop Node	o1	s1	Non-conformant final transition
A transition in the object life cycle is not covered by the process model.		o1	s1->s2	Non-covered transition

Exercise

We now demonstrate the inconsistency resolution support using our customer order processing example from Tutorial 1. In Tutorial 1, you used the object life cycle extraction and the 5-step Object Life Cycle Validation method to ensure that the state evolution of the **Order** business item was correctly implemented in the **Customer Order Handling** process model. Several iterations were required until the desired object life cycle for **Order** was extracted from the process model. On each of the iterations, you changed the process model manually and then repeated the object life cycle extraction.

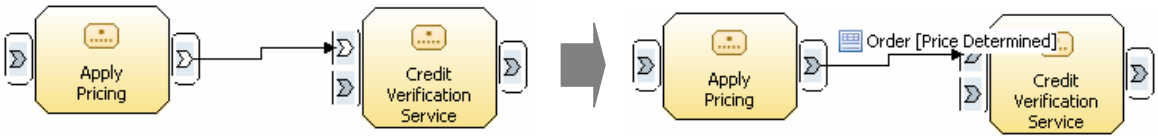
Here, we demonstrate an alternative approach to correcting the process model. Instead of changing the process model manually, you can edit the extracted object life cycle directly to represent the desired state evolution. Then, you can run the consistency check on the original process model and the edited object life cycle to detect the introduced inconsistencies and use the inconsistency resolution support to resolve them, thus automatically updating the process model.

Import the **Customer Order Handling** process (**Customer Order Handling.mar** in the **samples** directory) and the **Order Valid** object life cycle (**Order Valid OLC.zip** in the

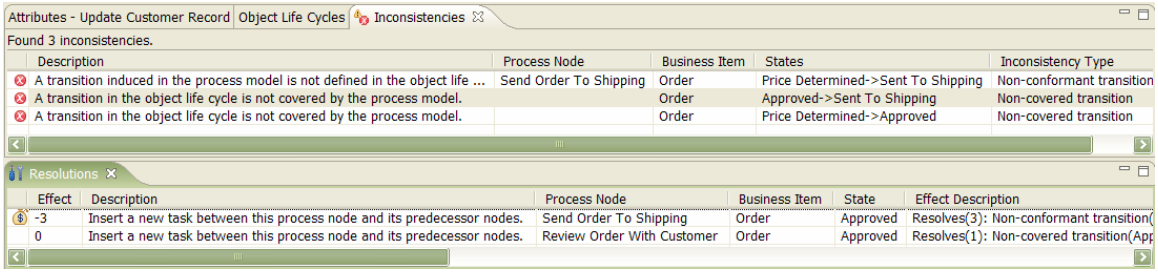
samples directory) into a new project. Open the **Customer Order Handling** process model and the **Order Valid** object life cycle. The **Customer Order Handling** process model is the same as the original one used in Tutorial 1, except that the **Credit Verification Service** has been converted from a global service to a task[2]. The **Order** life cycle represents the desired state evolution of the **Order** business item (it is the same as the life cycle obtained after manual corrections in Tutorial 1, shown on page 24).

Run the consistency check for the **Customer Order Handling** and the **Order Valid** object life cycle (right-click on a project element and select **Object Life Cycle → Check → Process/Object Life Cycle Consistency**). The Inconsistencies view is opened, showing 9 detected inconsistencies. For resolving these inconsistencies, it is convenient to use the layout in **Screenshot 1** shown on the following page: the process model and object life cycle side-by-side, the Inconsistencies and Resolutions views full-length at the bottom.

Select the non-conformant initial transition to state **Credit Check Negative**, as shown in **Screenshot 1**. In the Resolutions view, you can see that by associating the **Order** business item and the **Price Determined** state to the incoming connection of **Credit Verification Service**, 6 inconsistencies can be resolved at once. Before applying this resolution, navigate to the **Credit Verification Service** task in the process model editor and make sure that you are in the **Advanced** modeling mode. Right-click on this resolution and select Apply. The process model is changed as shown below, which is precisely how it was corrected manually in Tutorial 1.



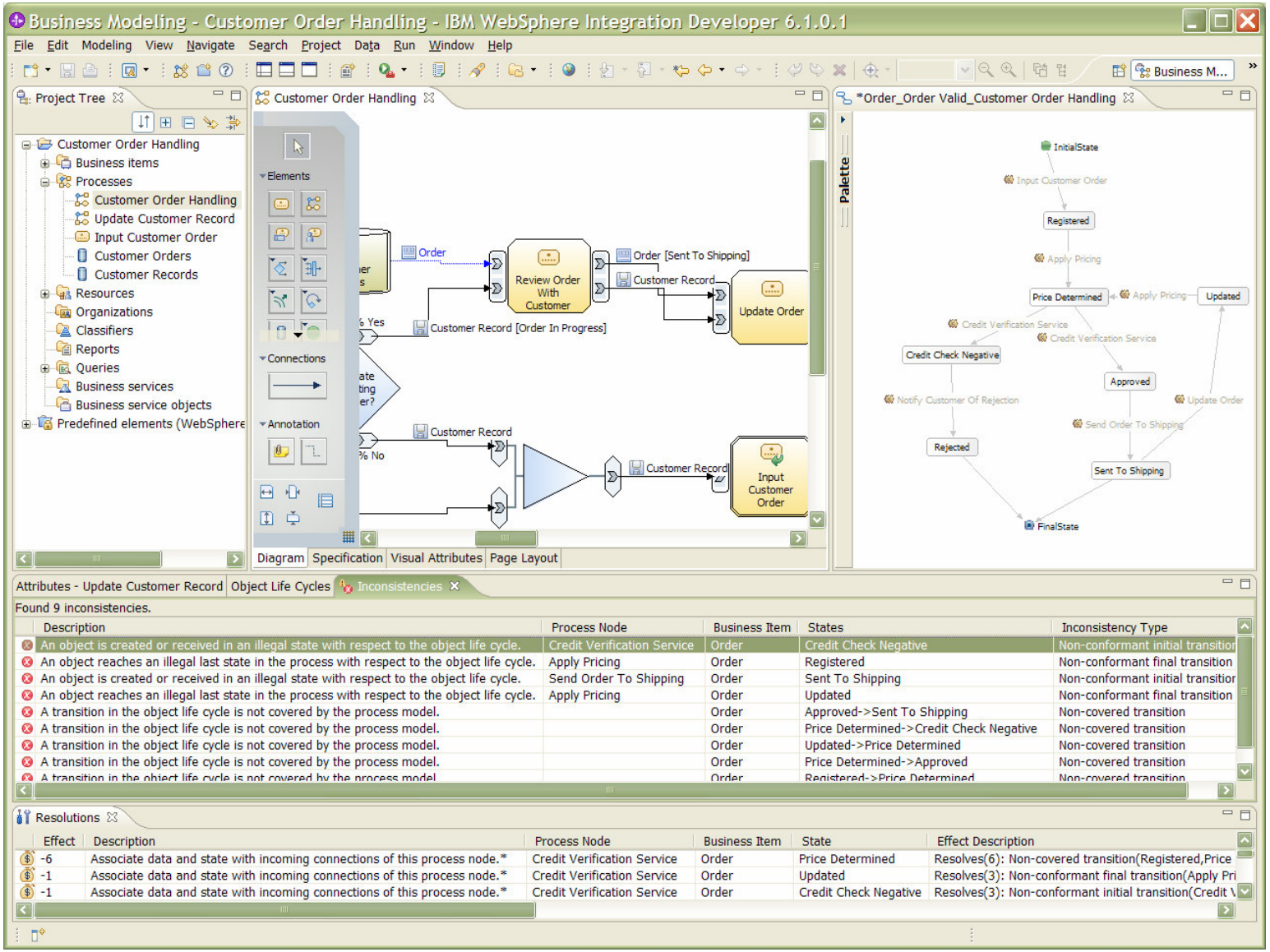
Note that the resolution that you just applied was marked with a star **(*)** in the **Description** column. This indicates that the consistency check may need to be repeated after the application of the resolution. Such resolutions are unable to always update the Inconsistencies view precisely, due to some complex structures in the process model[3]. Run the consistency check again to obtain the following results:



If you select the non-conformant transition from **Price Determined** to **Sent To Shipping**, you will see that no resolutions are applicable. This does not mean that it is impossible to resolve this inconsistency, as you can always resort to correcting the process model

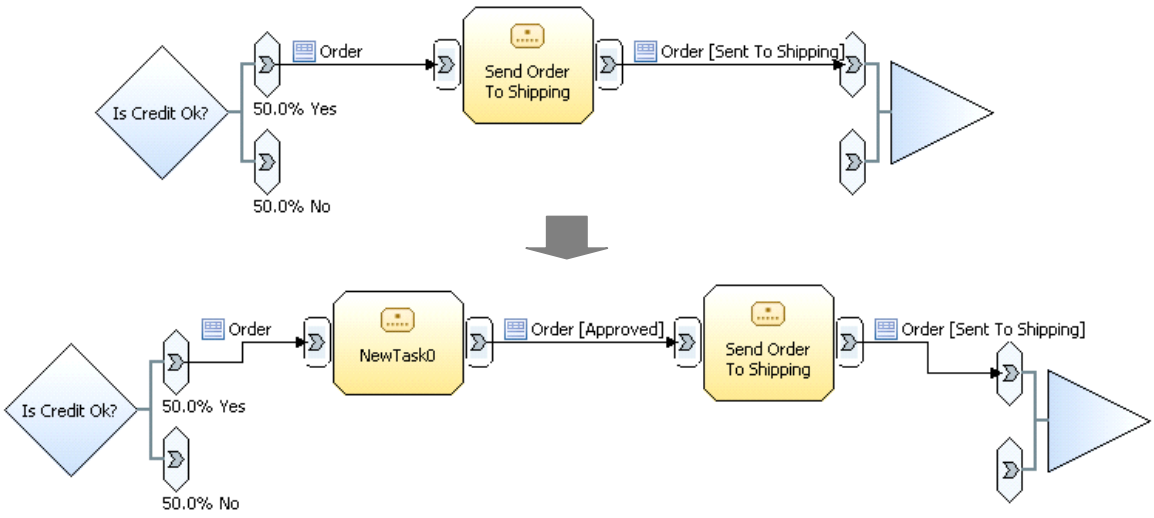
[2] Inconsistency resolutions currently do not fully support global tasks, processes and services.
[3] In this example, there is a decision following the **Credit Verification Service** with an incomplete state specification (only one outgoing decision branch has state specified for the **Order** business item).

Screenshot 1



manually. Of course, if you make manual changes, you will need to repeat the consistency check to ensure that the inconsistency was indeed resolved. However, we recommend that you first examine the other inconsistencies and attempt to resolve them automatically.

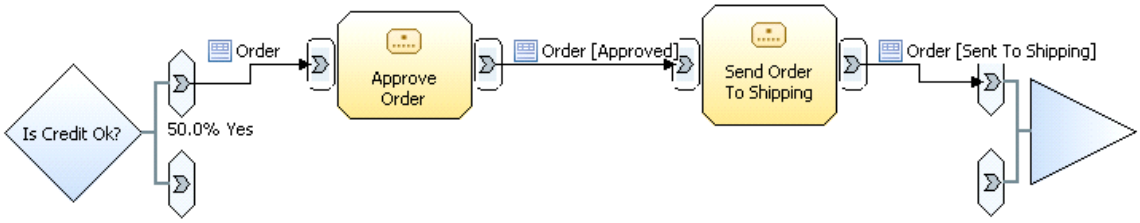
Select the non-covered transition from **Approved** to **Sent To Shipping** (as shown in the Inconsistencies view shown on page 55). This inconsistency can be resolved automatically and furthermore, the first resolution shown in the Resolutions view resolves not just 1 inconsistency, but all 3 inconsistencies. Apply this resolution. To ensure comprehensible formatting, apply the auto layout (right-click in the process model editor and select **Auto-Layout Left to Right**).



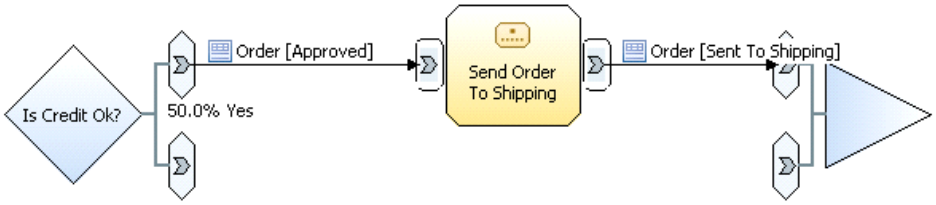
The newly inserted task can be subsequently renamed to **Approve Order**, for example.

In this way, the **Customer Order Handling** process model was aligned with the **Order** object life cycle that represented the valid state evolution for this business item. You should note that the resultant process model is not exactly the same as the one we obtained in Tutorial 1:

Process model obtained through automatic inconsistency resolution:



Process model obtained through manual changes in Tutorial 1:

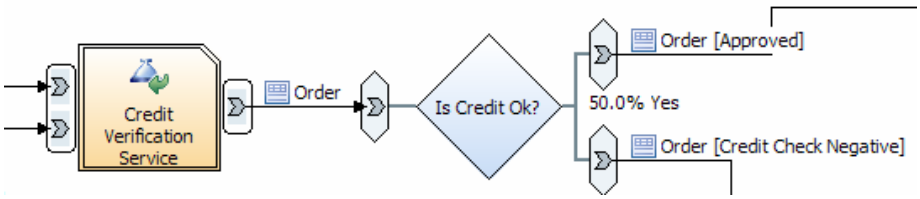


While both process models implement valid state evolution of the **Order** business item, one or the other may be more suited to the business or domain requirements. For example, if there is a need to perform some additional processing after the credit has been checked and before the order is sent to shipping, then the upper model that contains an **Approve Order** task represents the process more accurately than the lower one.

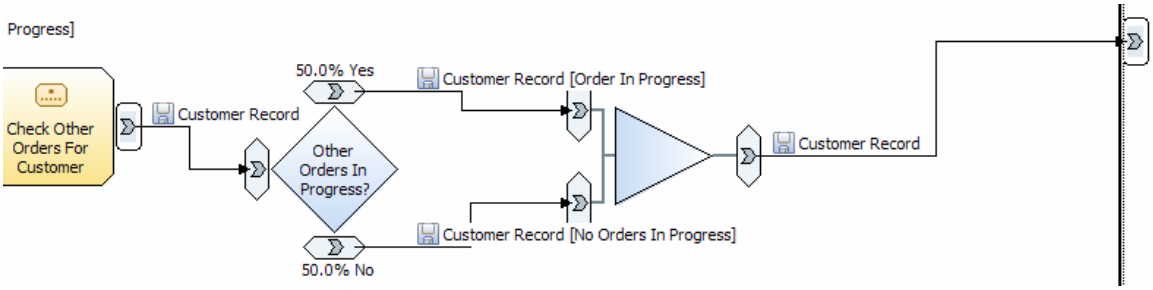
6. Advanced Topics

Specifying states using activity pre/postconditions

As you may have noticed, WebSphere Business Modeler currently does not allow you to directly specify multiple alternative input or output business item states for an activity. To model an activity with alternative input (output) states, you need to connect a merge (decision) to the activity and specify the alternative states on the input (output) branches of this merge (decision). Here is an example from Tutorial 1:



While following this modeling guideline may be adequate for most of your modeling, certain behaviors can only be modeled in a cumbersome way using this guideline. Another example from Tutorial 1 demonstrates this:

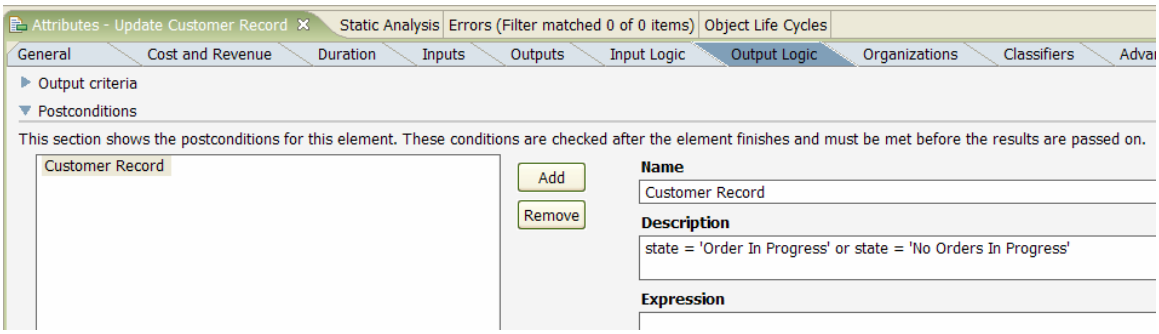


This is the **Update Customer Record** process, where the **Check Other Orders For Customer** task updates the state of the **Customer Record** business item either to **Order In Progress** or **No Orders In Progress**. Hence, the **Update Customer Record** process outputs the **Customer Record** in one of these two alternative states. It was necessary to model a decision followed by a merge to model this behavior, which clutters the process model.

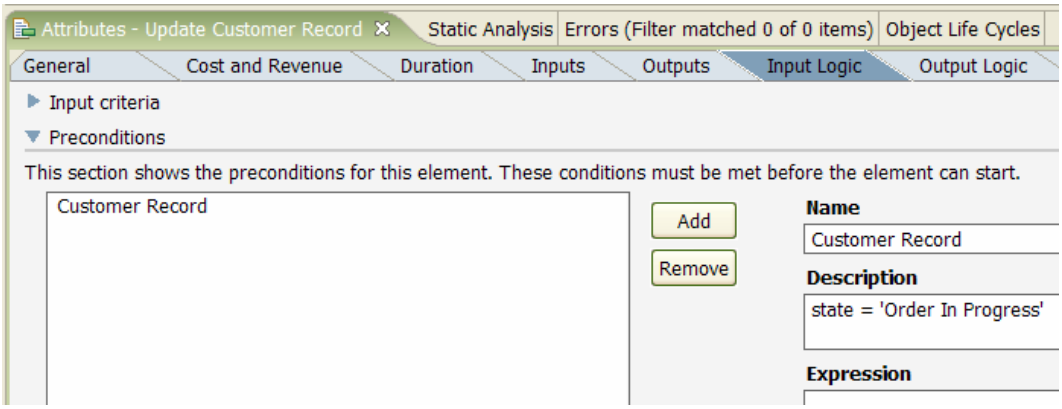
Object Life Cycle Explorer supports state pre/postconditions as another method of specifying business item states in process models to avoid such clutter in process models. Using this method, the state specification is not displayed in the process model editor, but is saved as part of the process model attributes.

Make sure that you are either in **Intermediate** or **Advanced** mode. Click on the empty space inside the **Update Customer Record** process model editor to open its attributes in the **Attributes** view. Select the **Output Logic** tab, open the **Postconditions**. Click on

Add to create a new postcondition. Rename it to **Customer Record** and fill out the **Description** field as shown below:

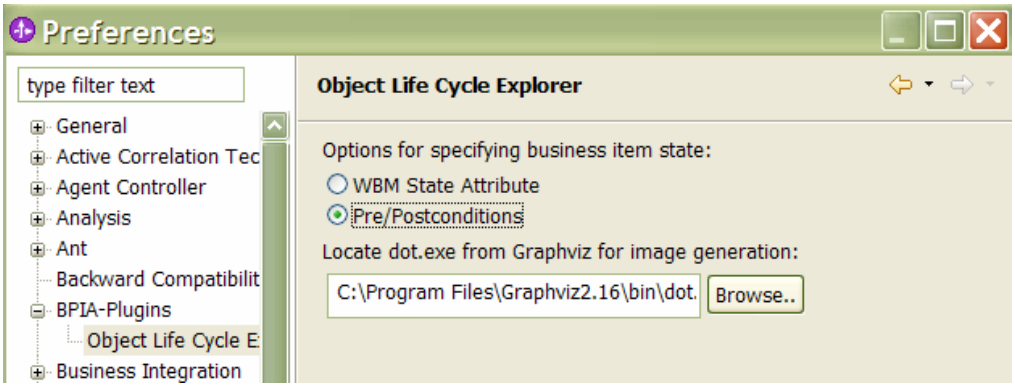


Similarly, you can add a precondition for this process to indicate that the **Customer Record** business item is in state **Order In Progress** when it is received via the input parameter.



Object Life Cycle Explorer features, such as the object life cycle extraction and the consistency check, are also available for the state pre/postconditions. You need to change a preference setting to enable this.

Go to **Window → Preferences**. Open the **BPIA-Plugins → Object Life Cycle Explorer** preference page. Under **Options for specifying business item state**, select **Pre/Postconditions** and click on **OK**.



Opening object life cycles in WebSphere Integration Developer as Business State Machines

Similar to the way you can export process models from WebSphere Business Modeler to Business Process Execution Language in WebSphere Integration Developer, one can obtain Business State Machines in WebSphere Integration Developer from the object life cycle models developed or extracted using Object Life Cycle Explorer.

The transfer to WebSphere Integration Developer is currently not supported automatically and requires the following steps to be performed manually:

1. Export the object life cycles from your project to an archive, e.g. **olcs.zip**.
2. Extract the exported archive into some temporary directory. Once extracted, you will find an **ObjectLifeCycle.registry** file and several **.sacl** and **.saclex** files**.**
3. Determine which files are used to store the particular object life cycle that you want to be transferred to a Business State Machine:
    - Open the **ObjectLifeCycle.registry** file in a text editor. Each line in this file corresponds to an object life cycle.
    - Locate the number corresponding to the object life cycle of interest by identifying the line that contains the business item name and the object life cycle name corresponding to your object life cycle. For example, this number is **0** if you are looking for the **Reference** life cycle for the **Payment** business item and there is a line "`0:Payment:Reference:0:`" in the **ObjectLifeCycle.registry** file.
    - Locate the **.sacl** file that corresponds to this number, e.g. **0.sacl**.
4. Open the WebSphere Integration Developer and create a new **Module**.
5. In the file system, copy the previously identified **.sacl** file storing the information about your object life cycle into the directory corresponding to your new module.
6. Open the **.sacl** file in a text editor and add the module information:
    a. Locate the second line in the **.sacl** file, e.g.:

```
<sacl:stateMachineDefinition
xmlns:sacl="http://www.ibm.com/xmlns/prod/websphere/wbi/sacl/6.0.0"
displayName="Payment_Reference" name="Payment_Reference">
```

    b. And change it as follows, where **module-name** is the name of the module created in WebSphere Integration Developer:

```
<sacl:stateMachineDefinition
xmlns:sacl="http://www.ibm.com/xmlns/prod/websphere/wbi/sacl/6.0.0"
displayName="Payment_Reference" name="Payment_Reference"
targetNamespace="http://module-name">
```

7. Refresh the module in WebSphere Integration Developer. A Business State Machine corresponding to the object life cycle should appear under **Business Logic → State Machines**.

8. Using this Business State Machine as a skeleton, refine it to a deployable component. One necessary refinement step involves resolving the operations associated with transitions to service interface operations.

How this technology works

Object Life Cycle Explorer for WebSphere Business Modeler implements several techniques for the integration of process and object life cycle modeling developed at the IBM Zurich Research Laboratory [3].

For the extraction of object life cycles from process models and the consistency check, automatic completion of the business item state specification is first performed in the given process models. States are propagated along object flows in the process models until a set of business item states is determined for every object input and output.

The object life cycle extraction is then performed by applying transformation rules, which map activities in process models to state transitions in object life cycles and identify initial and final states for each life cycle (see [4] for further reading).

The consistency check evaluates several consistency conditions on the given process models and object life cycles to detect non-conformant state changes of business items induced in the process models and parts of the given life cycles that are not covered by the process models (see [1, 2, 4] for further reading).

For the generation of a process model from a set of object life cycles, a composition of the life cycles is computed first and then transformation rules are applied to map the composite life cycle to a process model (see [1] for further reading).

7. Current Limitations

- Notification broadcasters / receivers, observers, timers and maps are not supported. Business item states specified for these elements is ignored during the object life cycle extraction and the consistency check.
- During the object life cycle extraction, information about the unique location of activities may be lost. For example, if there are two activities with the same name in different subprocesses of the same process model, these cannot be distinguished in the extracted object life cycles.
- Inconsistency resolution support currently has the following limitations:
    - Resolution is only supported between one process model and one object life cycle. When multiple process models or life cycles are selected during the consistency check, inconsistency resolution is disabled.
    - During the application of a resolution that adds a new state to an existing business item, the process editor is closed for synchronization with the new business item definition and then reopened again. The layout of your editors may change as a result.
    - Transformations associated with resolutions offer limited support for global tasks, processes and services. Certain resolutions are not applicable to resolve inconsistencies associated with such process nodes.
    - Resolution side-effects are not always computed precisely in the presence of complex process structures with an incomplete state specification. Such resolutions are marked with a star (*). After application of such resolutions, the consistency check should be repeated.

8. References

[1] J. M. Küster, K. Ryndina, H. Gall. Generation of Business Process Models for Object Life Cycle Compliance. In proceedings of the 5th International Conference on Business Process Management (BPM), volume 4714 of LNCS, pages 165 -181. Springer, 2007.

[2] J. M. Küster, K. Ryndina. Improving Inconsistency Resolution with Side-Effect Evaluation and Costs. In proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS), volume 4735 of LNCS, pages 136-150. Springer, 2007.

[3] K. Ryndina, J. M. Küster, H.Gall. A Tool for Integrating Object Life Cycle and Business Process Modeling. In proceedings of the BPM Demonstration Program at the 5th International Conference on Business Process Management (BPM). CEUR-WS, 2007.

[4] K. Ryndina, J. M. Küster, H.Gall. Consistency of Business Process Models and Object Life Cycles. In Workshops and Symposia at MoDELS 2006, volume 4364 of LNCS, pages 80-90, Springer, 2006.

Authors

Object Life Cycle Explorer was developed in the Business Integration Technologies group at the IBM Zurich Research Laboratory.

Ksenia (Ryndina) Wahler is a doctoral student at the IBM Zurich Research Laboratory; she is pursuing a Ph.D. in computer science at the University of Zurich. Ms. Wahler is the lead architect and developer of Object Life Cycle Explorer. This technology implements several novel techniques for integrating business process and object life cycle modeling, which were developed as part of Ms. Wahler's Ph.D. research.

Email: ryn@zurich.ibm.com

Jochen Kuester is a research staff member at the IBM Zurich Research Laboratory. He holds a Ph.D. in computer science from the University of Paderborn, Germany. Dr Kuester's research interests include business process modeling, model transformations, and model-driven development of service-oriented applications. He plays a major role in the conceptual development and design of the Object Life Cycle Explorer.

Email: jku@zurich.ibm.com

Aurelien Monot is an intern at the IBM Zurich Research Laboratory and a graduate student at the computer science department of the Ecole Nationale Superieure des Mines de Nancy. Mr. Monot contributes to the design, implementation and testing of Object Life Cycle Explorer as part of his final-year internship. His current research interests include model-driven development, model analysis, and real-time systems.

Email: mon@zurich.ibm.com

This release would not have been possible without the cooperation of other members of the Business Integration Technologies group and Grace Wong from the WebSphere Integration Developer team in IBM Software Group.