

ArgoUML and Poseidon for UML

Users Guide and Manual

ArgoUML and Poseidon for UML: Users Guide and Manual

Copyright © 2001 by Marko Boger, Gentleware

Copyright (c) by Marko Boger, Gentleware.

This material may be altered and distributed according to the terms and conditions set forth in the Open Publication License, v1.0 or later (latest version available at <http://www.opencontent.org/openpub/>).

Parts of this documentation originate from the ArgoUML Project. Special thanks to Kunle Odutola and Denny Daniels.

Table of Contents

About ArgoUML and Poseidon for UML.....	
1.Prerequisites.....	
2.Installation and first Start.....	
Start through Java Web Start.....	2
Local Installation.....	3
Start as Module in Netbeans or Forte for Java.....	4
3.Guided Tour.....	
Opening the Default Example.....	5
Introducing the Workareas.....	5
Navigation.....	7
Working with Diagrams.....	8
Arranging Diagram Layout.....	9
Creating new Diagrams.....	10
Drag and Drop, Copy/Cut/Paste.....	11
Removing and Deleting.....	11
Saving and Loading Projects.....	12
Exporting Graphics and Printing.....	13
A Walk through the Diagrams.....	13
Constraints with OCL.....	15
Critiques.....	15
Generating Code.....	15
Reverse-Engineering Code.....	16
Round-Trip Engineering.....	16
Searching Model Elements.....	16
4.Reference Manual.....	

About ArgoUML and Poseidon for UML

According to greek mythology, the hero Jason built a ship called Argo and with his comrades, the Argonauts, he left for the quest of the golden fleece. Poseidon, god of seas, protected and safely guided their journey.

About 4000 years later, Jason Robbins starts an open source project for a UML modeling tool and calls it *ArgoUML*. Many others join him in this adventurous undertaking and the outcome is what you find described in this document.

ArgoUML was conceived as a tool and environment for use in the analysis and design of object-oriented software systems. In this sense it is similar to many of the commercial CASE tools that are sold as tools for modelling software systems. ArgoUML has a number of very important distinctions from many of these tools:

- ArgoUML includes a number of features that support the cognitive needs object-oriented software designers and architects. This was one of the main incentives to start the ArgoUML project.
- ArgoUML supports open standards extensively - UML, XMI, SVG, OCL and others. In this respect, ArgoUML is still ahead of many commercial tools.
- ArgoUML is a 100% Pure Java application. This allows ArgoUML to run on all platforms for which a reliable port of the Java2 platform is available.
- ArgoUML is an open source product. The availability of the source permits ensures that a new generation of software designers and researchers now have a proven framework from which they can drive the development and evolution of CASE tools technologies.

Many features of the tool have evolved naturally in the open source project, but some more advanced topics require the full-time effort that only a commercial company can provide. Some of the core developer team have taken up the task and started the company *Gentleware*. They provide extensions in a tool suite called *Poseidon for UML* as well as commercial services around these tools.

The basic version of Gentleware's tool suite, *Poseidon for UML Community Edition*, is free of charge. It is based on ArgoUML and especially the user interface and general usage of both tools are mostly identical. Therefore this document describes both versions simultaneously. Poseidon for UML is more feature rich and more stable. It is intended for your daily work in commercial and professional environments. ArgoUML, on the other hand, is open source and lends itself for research, study of the architecture and for extensibility. It is assumed that Poseidon is the tool more frequently used and by a community more in need of a Users Guide and Manual. Therefore this document

normally refers to Poseidon for UML. Differences to ArgoUML should be made explicit.

Chapter 1. Prerequisites

Poseidon for UML is entirely written in Java and therefore platform independent. It runs on almost any modern personal computer. To successfully start and run Poseidon for UML you need the following:

- Java Runtime Environment or Java Developer Kit. JDK 1.3 is recommended. A version later than 1.2 is required. Poseidon for UML will not run with JDK 1.1.X or JRE 1.1.X.
- A computer with reasonable memory and CPU power. For memory, 128 KB is recommended, more is helpful. For CPU, a Pentium III or equivalent is recommended.
- A specific operating system is not required. Poseidon for UML is known to run on Windows 98, 2000 and NT, on Linux SuSe 6.X, 7.X and Red Hat as well as on MacOS X. It is mostly developed and most tested on Linux, however, on Windows platforms performance is known to be superior. Also, some implementations of the JDK for different platforms have bugs. Drag and Drop, for example, does not run under Linux/JDK 1.3

Chapter 2. Installation and first Start

This chapter informs you where you can get Poseidon for UML from, what you need to do to install it and how you start it.

To install Poseidon for UML, there are currently three possible ways. You can do either of the following:

- Download over the internet and locally install Poseidon for UML
- Install Java Web Start and start Poseidon for UML from the internet
- Install Netbeans or Forte for Java, download Poseidon for UML and install it as module for Netbeans or Forte for Java

Start through Java Web Start

Java Web Start is a mechanism provided by Sun Microsystems to automatically install and start applications from the internet. After Java Web Start is installed, all you need to do is double-click a provided link. The required files are then automatically loaded to a cache on your local disc and the program is started. The first time this may take a little while, but the second time around, most information is taken from the local cache.

The big advantage of this mechanism is, that if the program is updated on the server, you will automatically start the new version. This also works if you are not online. In that case the local copy from the cache is used.

First, Java Web Start needs to be installed. Follow the following steps:

1. Download the Java Web Start installation file. You can get it from
 - Gentleware at <http://www.gentleware.com>,
 - Tigris at <http://www.argouml.org>, or directly from
 - Sun at <http://java.sun.com/products/javawebstart/>

You will automatically be provided with a self-installing file for your platform.

2. Close all your browser windows

3. Execute the downloaded file.
4. Open your browser again. Go to <http://www.gentleware.com/products/> and click on the icon provided for Java Web Start. After a few moments, Poseidon for UML in the latest released version will start up automatically.

Local Installation

Local installation is very simple. In short, download the file, unzip it, open the created folder and run the start script. Follow the following steps:

1. To locally install Poseidon, you first need to download the according file over the internet. Make sure that you are connected to the internet. Then open your favorite internet browser and direct it to <http://www.gentleware.com>.
2. Navigate to the download area and follow the instructions. You will then have a single file stored on your local hard drive in a location you indicated.
 - This file is compressed using Zip. Move this file to a folder where you would like to install Poseidon for UML. Then, to uncompress it, call the zip-program used on your platform. Here are some examples:
 - On Linux or Unix, open a command shell, go to the folder were the downloaded file is stored using the **cd** command, and call **unzip PoseidonCE-1.0Beta2.zip**.
 - On Windows start the Zip program. If the Zip programme is installed properly, it should automatically start if you double-click the downloaded file. Then extract the file to a folder of your choice by clicking on the extract button and following the instructions.
3. Open the folder `poseidon`.
4. Run the start script that is provided for your platform.
 - On Linux or Unix, open a command shell, go to the folder `poseidon` and enter the command **startPoseidon.sh** command, and press return.
 - On Windows, open the file browser and double-click the start script **startPoseidon.bat**.

Start as Module in Netbeans or Forte for Java

Poseidon can be started as a module for the development environment Netbeans (open source) or Forte for Java. Of Netbeans a version 3.2 or higher is required. For Forte it needs to be 3.0 or higher.

To install it as a module, download and install Poseidon as described in the last section, then procede as follows:

1. Open the menu item Tools->Options.
2. Select Modules. Right-click and select new module.
3. Now direct the file chooser to the directory where you unzipped the Poseidon distribution, go to subfolder lib and select the file poseidon.jar.

A new workspace should appear that contains the panes used in Poseidon.

Chapter 3. Guided Tour

This chapter introduces all basic concepts of Poseidon for UML by leading you through an example. It is a guided tour that takes you to most of the features Poseidon for UML without explaining all details. It gradually teaches you what you can do with Poseidon for UML and how you can achieve that yourself. However, it is not a reference: it does not reveal all details.

Opening the Default Example

So let's start a guided tour through Poseidon for UML. Your distribution comes with a default example built in. This example is used for the guided tour.

To open the default example, do the following:

- Start Poseidon.
- In the main menu, select Help, then Open Default Example.

The project you opened is designed to teach you pretty much everything you need to know about UML and about Poseidon for UML. The example is taken from an e-commerce scenario where the company Softsale needs to model its business processes and create an according software. It shows a typical scenario how Poseidon for UML can be used. However, UML as well as Poseidon for UML are not restricted to this kind of application, it is a general tool to model any kind of object-oriented software system or even systems that have nothing to do with software at all. We'll learn more about the example later. When the project is fully loaded, the screen should look like this:

Introducing the Workareas

The working window of Poseidon is separated in four areas. The biggest area is called the *Diagram Pane* where usually the UML diagrams are displayed. To the left of it, you find the so called *Navigation Pane*. Let's start with those two.

Models in UML are organized in Packages. The main package used in this example is called `softsale`. It is displayed at the top of the navigation pane. You can open it by clicking on the tree icon in front of the package name. Then your navigation pane should look like this:

The package `softsale` contains several other packages (`clients`, `ordering`,

products, boni and java), as well as a number of diagrams (packageOverview, mainClassesOverview, ...) and model elements (Surfer, Member, ...). The diagram at the top of the list, the packageOverview, is currently displayed in the diagram pane that shows the dependencies between the packages clients, ordering, products and boni.

The navigation pane is the main mechanism for navigating through a UML model. You can navigate to all higher level model elements and diagrams from here. For example, open the second diagram, called mainClassesOverview by double-clicking on the icon in front of its name in the navigation pane. Then your diagram pane should display as follows:

This diagram already tells you quite a bit about what this example is all about. It models Clients, that are associated with an Address, have an Account, and might place Orders containing Products, that can be, for example, DigitalProducts. However, this is just an overview diagram and only provides a high level view.

Lets look at some details. For this, go to the package clients in the navigation pane and open the diagram clients.

Within the package clients, there are the classes Client, Account, Address, Clients and CreditCard and a few associations. To find out more information on the class Client, open the Client class in the navigation pane by clicking on the tree icon and then select the Client class by clicking either on the name in the navigation pane or on the according class element in the diagram.

Now, lets take a closer look at the window at the bottom, the *details pane*. This pane displays many detail informations about the element currently selected and allows to edit them. It consists of a number of different panels that can be selected through according tabs at the top edge of the pane. The most important panel in the details pane is the *property panel*. So far only packages and diagrams were selected and their properties are not very interesting. But now that a class is selected, it becomes an important tool to view and change the model details. For example, try to change the name of the class Client to Customer. Watch how the name changes in the diagram as well as in the navigation pane (well, sorry, we currently still have update problems here, but we'll fix that soon) as you type the new name in the property panel. Actually, the name is changed in the entire model, including the diagram mainClassesOverview that we looked at earlier. Yes, go check it out.

The fourth pane is called the *todo-pane*. It collects critiques and allows to sort them according to different criteria. Critiques are one of many features offering cognitive support to the developer. It is a very interesting feature, but it usually is needed a little later in the development process. So we'll get to that a little later.

Navigation

A UML model can become quite complex. There are various informations that are important to different people and at different times. A UML-tool should provide elaborated yet simple to use mechanisms to access that information. This is called navigation. Poseidon offers many different ways of navigating between the model elements. Here, we'll look at the most important ones.

The most central mechanism for navigation is the navigation pane that we already had a first look at. It organizes the UML model in a tree view and by opening and closing the subtrees it gives access to almost all elements of a model. However, you can not change model elements in the navigation pane and you can not access those elements at a very detailed level.

But let's look at the possibilities it directly offers first before we'll see how it interacts with the other windows. At the top edge of the navigation pane you find a drop-down-menu that currently displays “package-centric”. And indeed the content of the navigation pane is currently organized by the package structure: The top-level package is `softsale`, that contains other packages like `clients` that contain classes etc. You can change this to be centric to some other organization criteria. For example, the navigation pane can be organized in a diagram-centric way. Select diagram-centric in the drop-down menu and the navigation pane will change to display the following:

Now you can see all diagrams contained in the example model at one glance. By clicking on one of the diagram names or icons, the according diagram opens in the diagram pane and by opening the subtree the elements contained in that diagram are displayed. The first two are the most commonly used views, the others are for more special cases, for example to find out the inheritance structure of the model or the structure of the navigation paths. And remember, the navigation pane displays the complete model while a diagram might only show you some aspects. Select the `clients` class diagram, open its subtree and then select the `client` class in the navigation pane.

Each time you make a selection in the navigation pane, the details pane is updated and shows the details for that selection. Usually the details pane shows the property panel that contains the most relevant details. We'll look at that in a minute but check out some of the other panels now. If you select a different tab, the according panel will be displayed (as long as it makes sense for the current selection). For example, if you want to change the style of an element you can select the style tab and change the color in which it is displayed. Let's take the `client` class, a very centric class, and set its lines to bright red and its fill style to pink. You can also turn on and off the compartments for attributes and operations here, try to do so for the `Account` and `Address` classes so that you get the following display.

Now lets go back to the property panel. This is the place to drill down deeper into the model. It displays and allows to directly change the properties of an element. The `client` class for example is currently `public` and has the stereotype `entity`. Other properties are yet further down in the model and you need to navigate further to change them. Take a look at the some of fields in the properties panel, like `Associations`, `Operations` and `Attributes`. Everything that is marked as blue in these fields operates like a hyperlink in a hypertext would do. You can navigate from the class to its associations or operations and look at or change the detail information on those. Select the `checkPassword` operation and have a look at its properties. It is currently `public`, but you can change it to be `private` or `protected`. Note that while you do that the little icon in front of the operations name in the diagram pane changes from `+` to `#` and then to `-` accordingly.

Take a closer look at the parameters. They have properties themselves and thus have their own property panel you can navigate to. Note the parameter type `return`. The UML specification treats return types as special parameters. Thus every operation has a return parameter that by default is set to `void`. In this example, though, the return type of `checkPassword` is set to `boolean`. With the `back`, `forward` and `up` button you can comfortably navigate back and forth through the properties panes, also similar to a hypertext browser.

Working with Diagrams

Lets take a closer look at the diagrams pane. At the top edge of the diagram pane, you find a number of tools you need to create and modify your UML models. If you have ever worked with a UML tool or even just a drawing tool capable of creating UML diagrams, you are probably familiar with the general idea of such a tool bar. Each diagram type has a specialized set of tools in its toolbar. Some of them are common to all, though. Here is the tool bar for a class diagram:

If you want to find out the name of each individual tool, position your mouse over it an wait a little while, then the name will appear just underneath the tool. The first is called the *Select* tool. It is the default tool and is used to select diagram elements, move them around and scale them. If this tool is active (it is if you did not select a different one) you can select an element, for example the `client` class, and you will see that small blue handles appear on the corners of the element. You can use these handle to resize it. A selected element can be moved around if you select it again and keep the mouse pressed.

The second tool is called the *Broom*. It can be helpful to arrange your diagram, especially for horizontal and vertical alignment. We'll look at that in a minute. These first two tools are common to all diagram types.

3. Guided Tour

The next set of tools are specialized for each diagram type. They allow the creation of diagram elements and operate similar to a stamp. With a single click on such a tool you can create one according diagram element. If you double-click, you can create a number of diagram elements, one after the other. The cursor changes to a hair cross with which you can select the position where to create the element in the diagram. In the example of the class diagram, the first tool creates a package, the second a class. The next tree can be used to define associations between classes or packages. For these to operate, you already need to have the classes (or packages) you want to connect in place. Place the mouse over the first, click and hold, move the mouse to the second and release the mouse.

Some tools are only available in a certain context. In the class diagram for example, the tool to create a new attribute or a new operation are only available when a class is selected. If that is so, simply clicking on the button will create a new attribute or operation for that class, respectively.

The last set of tools are for general drawing purposes and are again common to all diagram types. With these you can add other graphical elements to your diagram. You should keep in mind, though, that these are not part of UML.

But the tool bar is not the only way to create new diagram elements or associations. Poseidon provides an intelligent shortcut that can speed up the development of a diagram quite a bit. Select a class (and move your mouse just a little bit) and you will see additional buttons appear just off the edges of the class. These buttons are called *Rapid Buttons* and are only available if an element is selected and the mouse is over it. They are available for many diagram elements and allow to quickly create the most important new elements that can be associated with it or to create a new association to an existing element. For a class these are associations to other classes and inheritance from a superclass or to a subclass. Try to click on the rapid button underneath a class and you will see that a new subclass appears where Poseidon thinks is a good place to put it. If you click and hold the button, you can move the mouse and place the new element where you like it to be. Or if you click, hold and move it over an existing element, only the new association between these elements is created. Note that the rapid button just over the class creates a superclass.

Arranging Diagram Layout

You already saw that you can lay out your diagram by using the select tool and moving elements around. There are a number of other ways to beautify your diagrams. You can not only move a selected class around by using the mouse but also by using the arrow keys of your keyboard. Your elements move in the direction of the arrow in small increments. Now hold down the shift key while pressing the key arrows. You will notice that the elements move in larger increments. You can select several elements by hold-

ing the shift key while selecting other elements. Now movements apply to all selections. You can also select a number of elements by clicking somewhere on the workspace, hold the key and drag the mouse. A blue rectangle appears and all elements that are completely enclosed in it will be selected.

If you want to select all elements of a diagram, you can use a hot-key to do that: press CTRL-A.

But there are still other options. Select the tree classes `Email`, `Client`, and `CreditCard`. Now go to the main menu and select `Arrange->Align`.

Select `Align Tops`, then select `Arrange->Distribute->Distribute Horizontal Spacing`. Then the outcome should look like this:

All three classes aligned on the top edge and equal horizontal distance between them. The other submenu-items of `Arrange` should now be self explaining. If not, go try them out.

The layouting is supported by a grid. If you want a finer or a coarser grid than the default or if you want the grid to be displayed in a different manner, you can change this in the view menu.

You can also change the layout of the edges. By default, Poseidon always tries to draw a straight line without bendpoints but you can easily add bendpoints: Select an edge and move it sideways. At first the edge simply moves sideways, too. But as soon as a straight edge is no longer possible, a bendpoint is automatically added. You can add several bendpoints to an edge so that you can wire your diagrams just as you like it. To remove a bendpoint, just move it exactly over a different bendpoint and it is gone.

You can also move annotations around. Simply select the annotation and drag it around. You will notice a little dotted red line that indicates to which association this annotation belongs.

Creating new Diagrams

So far we have only been playing around with the class diagram. But Poseidon supports all diagrams of UML. You can look at some examples by clicking your way through the example file. But how can you create your own diagrams?

Creating Diagrams is simple, but you might need a few informations to get it right. In UML, most diagrams are assigned to a package. When creating a new diagram, it is assumed you want it to be placed in the package you are currently working with. So if you want it to be in a specific package, select that package first in the navigation pane. Then, select `Create` from the main menu. From there you can select the diagram type you want.

3. Guided Tour

If what you wanted was a state diagram or an activity diagram, you might have found that they are greyed out in the menu. The reason for this is, that these diagrams are not directly assigned to a package but to a class or a use case. A State Diagram is always assigned to a class, an Activity Diagram can be assigned to either of them. So to create these diagram types, first select a class or a use case in the navigation or diagram pane.

If what you wanted was an Object Diagram or a Component Diagram, don't worry, they are there, too. They just don't have their own editor. Select a Deployment Diagram, it contains everything you need for all three of these diagram types. Using just one editor for all three diagram types allows to combine these diagrams.

You can give your new diagram a name by selecting it in the navigation pane and changing the default name in the property panel.

Be careful with diagram creation, though. The deletion of diagrams is a feature that still is not implemented. To follow along the guided tour, create a new class diagram for the top-level package `softsale`.

Drag and Drop, Copy/Cut/Paste

You have just created a new diagram. You can now fill this diagram either with new elements using the tool bar and the rapid buttons as we have just seen. But maybe you want to reuse the elements already present in the model, you just want to add a new presentation for them. For this you can drag existing elements from the navigation pane and drop them in the diagram. These elements will appear with all currently known associations to other elements already present in the diagram. For example drag the classes `Client`, `Address` and `CreditCard` to your new diagram. Your diagram should look something like this:

The other possibility is to select elements in a different diagram, copy them by pressing CTRL-C and pasting them into your new diagram by hitting CTRL-V. To cut element from a diagram, use CTRL-X. Of course you can also use these features from the edit menu.

Removing and Deleting

Now that you have created different views of one model element in different diagrams, you are well prepared to learn the difference between removing an element from a diagram and deleting it from a model. If you are used to working with drawing tools like Visio or Powerpoint to draw your UML-diagrams, deleting an element from your diagrams simply removes the according figure. With proper UML-modeling tools this is different. You are working on a model and the diagrams are just rendered from these.

This implies that there are two different meanings of “deleting an element”. With the *delete* key, an element is completely removed from the model. That means, that it will be removed from the current as well as from all other diagrams and you can not get it back again. With *CTRL-R* you can remove it just from the diagram you are currently working with but it remains untouched in other diagrams and it also remains in the navigation pane.

Saving and Loading Projects

By now you might have made quite a few changes to the model. You may want to save your changes to permanent store in a file. Of course you can save projects in Poseidon, but there is something more to say about this. Poseidon does not save models in yet another proprietary format but it make use of a standardized saving format that is based on XML. The UML specification that is standardized by the OMG has provided a mechanism how to exchange models in between different tools. This mechanism is based on a special application of XML called XMI (which stands for XML Metadata Interchange). In theory this sounds very nice, you can save a model in one tool and load it again in another one. But in practice, the standard is just not standardized enough. Not only are there different versions of UML, but also of XMI. And then the interpretation of XMI also differs between tools. More importantly, though, the layout information of diagrams in not transported via this mechanism. So all you can interchange between tools is the information presented in the navigation pane.

Many other tools can import or export XMI. And with some the interchange with Poseidon works quite well. To our knowledge, though, Poseidon and ArgoUML are the only tools using XMI as standard saving format. This means that the diagram information has to be stored in additional files. This is done using a format called PGML, which is a predecessor to SVG, the Scalable Vector Graphics format, standardized by the W3C. There is one such PGML file for each diagram. Also, some additional information about the project needs to be stored, which is done in jet another format with the ending *.argo*.

As a user you don't have to worry about all this. All of these files are zipped together to just one compressed file with the ending *.zargo*. This is a regular ZIP file and you can uncompress it using a ZIP tool or the Java JAR tool if you want to have a closer look inside. You can even load the unzipped format (select the *.argo* file) but it is not recommended.

You can access saving and loading from the File menu just like in any other graphical tool. Equally you can save a project under a new name, using the Save as.. menu item. Or you can use the hot key *CTRL-S* to save a project. To create a fresh project, select New Project. To import an XMI file, change the Open Project file chooser to XMI and select the xmi-file.

Exporting Graphics and Printing

Another option you have from the file menu is to export graphics. If you want to use your diagrams in other documents, in a report or a website or a slide show, you can export them in a set of different formats. The formats currently available are GIF, Postscript (ps), Extended Postscript (eps) and Scalable Vector Graphics (SVG). For web content, use the first. For text documents use ps or eps. This works especially well with LaTeX documents. You can also use this format for desktop publishing tools as provided by Corel or Adobe. You can use either in regular text processing systems like MS Word or in presentation tools as Powerpoint. The most promising format, though, is SVG. Currently there are not many applications supporting it for it is still brand new. But in the future, this will be the format of choice for web content as well as for text documents or even to manipulate these diagrams in graphic tools. Currently, there is a plug-in from Adobe for the Internet Explorer, if you want to try to export diagrams in SVG and view them in a browser. Also there is a graphics tool available from the Apache project, called Batik.

But for now we have to make due with traditional formats due to the lack of applications. To save a diagram, select it in the navigation pane, then select Save Graphics ... from the File menu. You can also directly print a diagram to a printer. The hot key for this is CTRL-P.

The following section shows a number of GIF images exported from Poseidon, showing various examples of diagram types available in UML.

A Walk through the Diagrams

There is a lot to say about when to use which diagram type and what the role of it should be. That is what is referred to as Process or Method. But this is not the place to do so. Here we'll just look at some examples of the various diagram types and explain a bit more about the example application.

The first diagram to look at is the *Use Case Diagram*. Here users, or better said, the roles the users have towards a (software) system, the tasks or Use Cases they are involved in and the relation between the Use Cases are expressed. It is often used in early stages of the process to collect the requirements. Please note that a complete Use Case is not just the bubble that is used to represent it in the diagram, but a description of the Use Case, a typical scenario, exceptional cases, preconditions etc., mostly expressed in external text or on cards. It can also comprise of other diagrams like a Sequence Diagram or Activity Diagram that explain the Use Case scenario.

To analyze the application domain and to pin down the terminology used (or to be

used), Class Diagrams are applied. In this stage they remain relatively simple since they are usually used to discuss things with the domain expert who can not be expected to have computer technology background. The following *Class Diagram* is a typical example for this.

Then the business process can be modeled, usually using Activity Diagrams. The following example shows an *Activity Diagram* that depicts the rules and the process of paying an order. In the case of this example, this is not so simple, for Softsale will not accept an order if you have overdue payments open, will only allow payment by invoice if your e-mail and home address have been verified and some other rules. Take a closer look yourself.

Business process models do not lend themselves to implementation in an object-oriented way. If you want to go the UML-way you should brake the business process down and express it in terms of states for each object involved in the process. The following *State Diagrams* show a few examples of this.

Client State Diagram

CreditCard State Diagram

Account State Diagram

OrderController State Diagram

To express single scenarios or examples of these business processes and how the individual objects interact with each other during execution, you can depict that in a *Sequence Diagram*. The following one shows an example.

In parallel to the above the overall architecture needs to be developed, again using *Class Diagrams*. But now not only terms of the domain but also implementation specific stuff gets expressed. A general pattern of an architecture is the Model-View-Controller Pattern, or the *Boundary-Control-Entity-Schema*, as it is often rephrased in the UML community. According to this, the architecture is constructed in three layers. The first, the Boundary, is responsible for representing information to the user and to receive his interactions. The next layer, Control, contains the rules how to combine information and how to deal with interaction. And finally the Entity-layer holds the data and is responsible for its persistence. To which layer a class belongs is expressed using stereotypes. For Boundary, Control and Entity according stereotypes are available. An example for this is shown below.

After a while, clusters of classes that strongly interact and form a unit will start to peel out in the architecture. To express this, the according clusters can be represented as components. If taken far enough, this can lead to a highly reusable component architecture. But such an architecture is hard to design from scratch and usually evolves

over time. An example for a *Component Diagram* is shown below but this diagram type still needs to be further developed. So be careful when using this diagram type.

Finally the way the individual components are deployed to a hardware system can be described using the *Deployment Diagram*, also available in Poseidon.

Constraints with OCL

Generally speaking, UML is a graphical language. A graphical language is very suitable of expressing relatively high-level abstractions for architectures, workflows, processes etc. But for expressing very detailed and fine-grained things like algorithms, equations or constraints, textual languages might just be more suitable. The UML recognizes this and comes with a supplementary textual language to express constraints. This language is called the OCL, or Object Constraint Language for long. Since it is text it is simple to support by tools and many UML tools do just that and you can enter a line of text in fields reserved for constraints. But also because it is text, it is quite difficult to tell — just by looking at it — whether syntax and semantics are used correctly. Tools can very much help with this, but most tools just don't. Poseidon does, and to our knowledge is best at doing this.

Critiques

Before we get to generating code, let's check if our model is as nice as it should be. In fact there is a very nice feature of Poseidon that supports this: critiques.

Generating Code

UML wouldn't be worth much if all you get is pretty pictures. What you want when analyzing and designing a software system is the code. Poseidon provides a very powerful and flexible code generation framework based on a template mechanism. It is currently only used to generate Java code, but it is flexible enough to generate any kind of programming language, or even other languages like HTML or XML.

Code generation is usually based on the classes of a model (all information that could be displayed in a class diagram). The code can be generated to files using the menu Generation. But in fact, the code generation mechanism is fast enough to constantly generate it online while you are browsing through and changing your model. Take a look at the details pane at the bottom of your window. There is a tab named source. Select it and what you will see is the code that is generated for the class that is currently selected. This code is generated every time you make a new selection. So if you change the name of a class and you want to see the new name in the source panel, sim-

ply deselect and select the according class again. You will find the complete interface definition for the class, including package information, import statements, class definition, inheritance information, attributes and methods with visibility, type and parameters. As you would expect. Lets look at a simple example.

But Poseidon can do even more for you. It also generates setter and getter methods for public fields. For bidirectional associations to other classes, also the code to manage this relation is generated. Bidirectional associations in UML (associations with navigation in both directions allowed, which is default) need to be transformed to two unidirectional associations in most common programming languages. If one of these is set, the other should be set accordingly. Poseidon generates the according code for you. Here is an example.

Reverse-Engineering Code

Software engineers often run into the problem of having to re-engineer an existing project for which only code but no models are available. This is where reverse-engineering comes into play. This means that a tool analyses the code and auto-generates a model and a set of class diagrams for it. Poseidon can do this for Java code for which the source code is available and in a state where it can be compiled without error. To launch this process, go to the menu and direct the file chooser to the root package. It will then analyze this as well as all subsequent packages. The outcome is a model containing the according packages, all classes and their complete interface, as well as one class diagram for each package. Classes that are needed to make the model complete but are not present in the package structure are also established in separate packages.

Round-Trip Engineering

Code generation and reverse-engineering that code still does not make a round-trip engineering. Reverse-engineering generates a new model from existing code, but it does not by itself reconnect the existing code to an existing model. This is a feature that is still not implemented in Poseidon. But a good code generation and a good reverse-engineering mechanism are the main ingredients to bake such a mechanism. So stay tuned.

Searching Model Elements

When your models start to grow, you will want a nice mechanism to search for elements. Poseidon offers a powerful search tool that is not just based on text but on model information. It allows you to look for specific types of elements. The search tool is invoked from the menu or by pressing. Type in the name of the element you are

3. Guided Tour

looking for (you can also use * as place holder) and specify the type in which to look for. If you are looking for a class, this type would be MClass.

For each search, a new tab is created so that you can access older search results. You can also restrict the search space to be the result of an earlier search.

Chapter 4. Reference Manual